Report No. DOT/FAA/RD-94/6

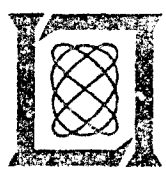
Project Report
ATC-213

The Polygon-Ellipse Method of Data Compression of Weather Maps

J.L. Gertz

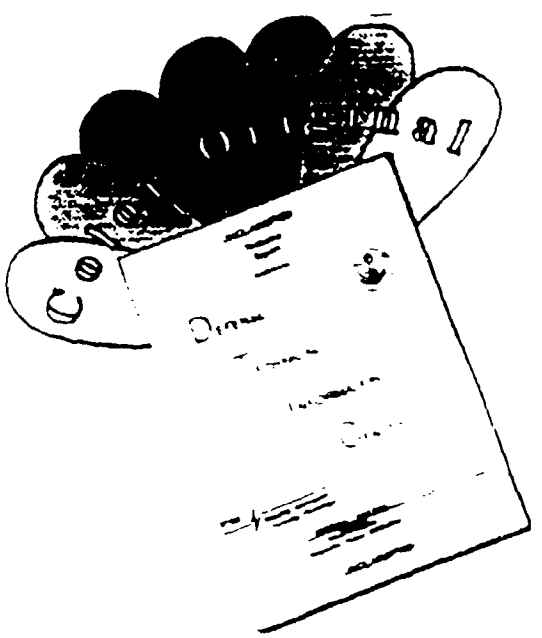
28 March 1994

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS



Prepared for the Federal Aviation Administration.
Document is available to the public through
the National Technical Information Service,
Springfield, Virginia 22161.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
ATC-213	DOT/FAA/RD-94/6	

4. Title and Subtitle	5. Report Date
The Polygon-Ellipse Method of Data Compression of Weather Maps	28 March 1994
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Jeffrey L. Gertz	ATC-213

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Lincoln Laboratory, MIT P.O. Box 73 Lexington, MA 02173-9108	11. Contract or Grant No. DTFA01-91-Z-02012

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Department of Transportation Federal Aviation Administration Systems Research and Development Service Washington, DC 20591	Project Report
	14. Sponsoring Agency Code

15. Supplementary Notes

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Air Force under Contract F19628-90-C-0002.

16. Abstract

Providing an accurate picture of the weather conditions in the pilot's area of interest could be a highly useful application for ground-to-air data links. The problem with using data links to transmit weather pictures is the large number of bits required to exactly specify a weather image. To make transmission of weather maps practical, a means must be found to compress this image.

The Polygon-Ellipse (PE) encoding algorithm developed in this report represents weather regions as ellipses, polygons, and exact patterns. The actual ellipse and polygon parameters are encoded and transmitted; the decoder algorithm redraws the shape from their encoded parameter values and fills in the included weather pixels. Special coding techniques are used in PE to compress the encoding of the shape parameters to achieve further overall compression.

The PE algorithm contains procedures for gracefully degrading the fidelity of the transmitted image when necessary to meet a specified bit limit. Pictorial examples of the operation of this algorithm on both Terminal Doppler Weather Radar (TDWR) and ASR-9 radar-generated weather images are presented.

Original contains color plates: All DTIC reproductions will be in black and white

Availability Codes	
Dist	Avail and/or Special
A-1	

17. Key Words	18. Distribution Statement
Data Compression Weather Images Weather Transmission Data Link	This document is available to the public through the National Technical Information Service, Springfield, VA 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	98	

FORM DOT F 1700.7 (8-72) Reproduction of completed page authorized

EXECUTIVE SUMMARY

It is often said that "a picture is worth a thousand words." This is certainly the case when it comes to giving the pilot of an aircraft a description of the weather. Providing an accurate picture of the weather conditions in the pilot's area of interest could be a highly useful application of a ground-to-air data link. Even those aircraft which have onboard weather radars could benefit from the ability to see the weather in regions beyond the range of their radar, or to see the nearby region with the greater clarity that a ground-based weather radar can provide.

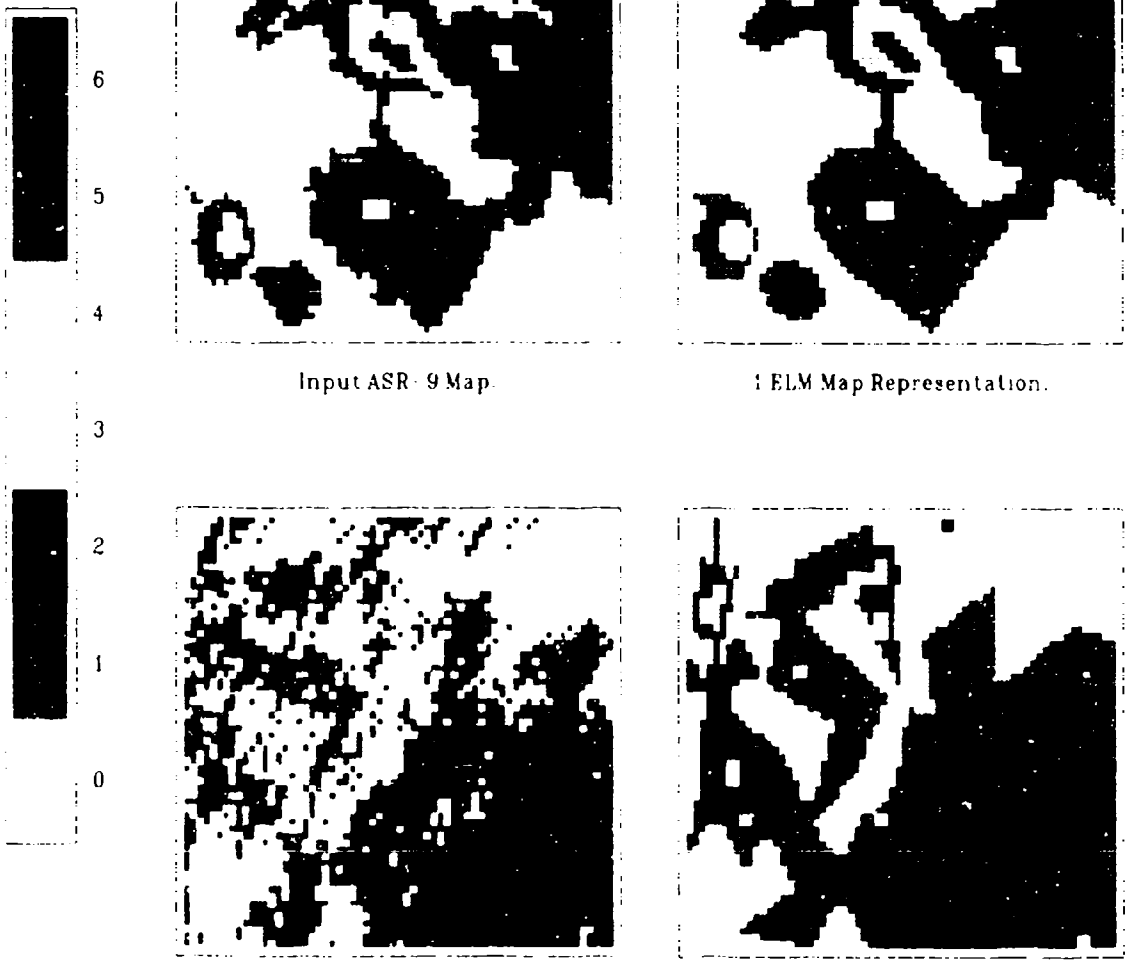
The problem with using a data link to transmit weather pictures is the need to transmit the large number of bits required to specify a weather image. For the purposes of this report, a weather map is defined as an array of 64x64 (4096) pixels. Such map granularity is believed to provide sufficient information for the pilot. Each pixel can indicate one of the seven National Weather Service weather levels and thus requires 3 bits to specify. Therefore, a weather-map image will consist of 12,288 bits of information, not counting any data link overhead. Unfortunately, the Mode S data link protocols, for example, can only guarantee the transmission of one 1280-bit ELM per aircraft per scan; other data links may be similarly limited.

To make transmission of weather maps using data links practical, a means must be found to severely compress the image. For example, Mode S would require approximately a 10-fold compression. In addition, the algorithms used to perform the compression and decompression must be computationally efficient: the ground computer may have to handle many aircraft requests in a given scan, and the airborne computer may neither be particularly powerful nor have extensive memory capacity.

The Polygon-Ellipse (PE) encoding algorithm developed in this report represents weather regions as ellipses, polygons, and exact patterns. The actual ellipse and polygon parameters are encoded and transmitted; the decoder algorithm redraws the shapes from their encoded parameter values and fills in the included weather pixels. The PE algorithm achieves high levels of compression because most weather regions are roughly elliptical or polygonal, and it generally takes fewer bits to specify the shape parameters of the region than to specify each of the pixels making up the region. In addition, special coding techniques are used in PE to compress the encoding of the shape parameters to achieve further overall compression.

Weather regions which are too large to be exactly encoded are considered for ellipse representation. A measure of the weather region's "ellipseticity" is made using the moments of inertia of the pixels making up the region. If the region's shape is sufficiently elliptical (within a parameter), it is encoded as an ellipse. The pixel locations of the two foci of the best matching ellipse and the distance parameter of that ellipse are calculated for transmission. Special techniques have been developed to reduce the number of bits to represent an ellipse from the nominal value of 33.

If a weather region is not sufficiently elliptical, it is fit with a polygon. The number of polygon vertices allocated to each weather region is a function of the map complexity. If the message-bit limitation is being exceeded, the number of vertices is reduced until the error area of the polygon relative to the true weather region reaches a parametric value. The polygon encoding is simply an ordered list of the polygon vertices. A 64x64-pixel map nominally requires 12 bits per vertex point (plus a vertex count). Again, special techniques often significantly reduce this requirement.



Examples of Compressed Weather Maps.

The PE algorithm requires almost 5.5 megabytes of memory to execute its compression procedure and about 50 kilobytes to execute its decompression procedure. This includes all code, data areas, and C libraries used. The large amount of memory used for PE compression is mostly composed of the precomputed data tables and save-areas used to increase performance; this requirement could be reduced by a factor of ten if speed were not critical.

The PE algorithm was tested on a set of 12 severe weather maps derived from data supplied by the ASR-9 and TDWR radars. Timing of the procedures was done with the VAX system clock, accurate to a 10-millisecond quantization. All testing assumed a 1-ELM (1280 bit) limit. The PE algorithm, as presently coded and run, required approximately 0.5 seconds on average to encode these severe maps (tests on more typical weather images produced results only one-tenth as long). The decoding routines, on the other hand, required, at most, 0.03 seconds for any complexity of map.

It is clear that the processing requirements for the airborne data link computer to perform decoding for the PE algorithm are quite reasonable:

onboard memory: 50 kilobytes

onboard processing: 0.03 seconds

The speed-optimized PE algorithm requires more computer resources to do its encoding procedure, but its requirements are still reasonable for modern ground-based computer systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

It is often said that "a picture is worth a thousand words." This is certainly the case when it comes to giving the pilot of an aircraft a description of the weather. Providing an accurate picture of the weather conditions in the pilot's area of interest could be a highly useful application of a ground-to-air data link. Even those aircraft which have onboard weather radars could benefit from the ability to see the weather in regions beyond the range of their radar, or to see the nearby region with the greater clarity that a ground-based weather radar can provide.

1.1 WEATHER-RADAR INFORMATION

The present TDWR (Terminal Doppler Weather Radar), ASR-9 (Airport Surveillance Radar), and NEXRAD (NEXt generation weather RADar) radars are designed to provide hazardous weather information to controllers located at the tower or at an en route center. While the voice messages that pilots receive from the terminal controllers will facilitate the avoidance of hazardous weather, they will not provide tactical "escape" information, leaving the choice of alternate flight paths to the pilot. With such a system, it is possible that the pilot may encounter weather situations as dangerous as those in the warning.

During the en route portions of flight, graphical weather information could be used in a number of applications to increase flight safety. For many aircraft, weather maps transmitted in enroute airspace would represent a unique source of information unavailable from any other source. These maps would be useful to warn the pilot of immediate weather hazards once encountered and to provide information of possible escape maneuvers. These maps could also provide long-range flight-path-planning information in a strategic sense.

1.2 MODE S DATA LINK

To provide pilots with the needed information to make informed decisions on the avoidance of hazardous weather, and to supply them with the same information the controllers have, the FAA is actively developing the capability to provide real-time graphical information of hazardous weather conditions to aircraft by use of the Mode S data link "Extended Length Message" (ELM) capability [1].

The problem with using a data link to transmit weather pictures is the need to transmit the large number of bits required to specify a weather image. For the purposes of this report, a weather map is defined as an array of 64x64 (4096) pixels. Such map granularity is believed to provide sufficient information for the pilot. Each pixel can indicate one of the seven National Weather Service weather levels (see Table 1) and thus requires 3 bits to specify. Therefore, a weather-map image, such as the examples shown in Figure 1-1, will consist of 12,288 bits of information, not counting any data link overhead. Unfortunately, the Mode S data link protocols, for example, can only guarantee the transmission of one 1280-bit ELM per aircraft per scan; other data links may be similarly limited.

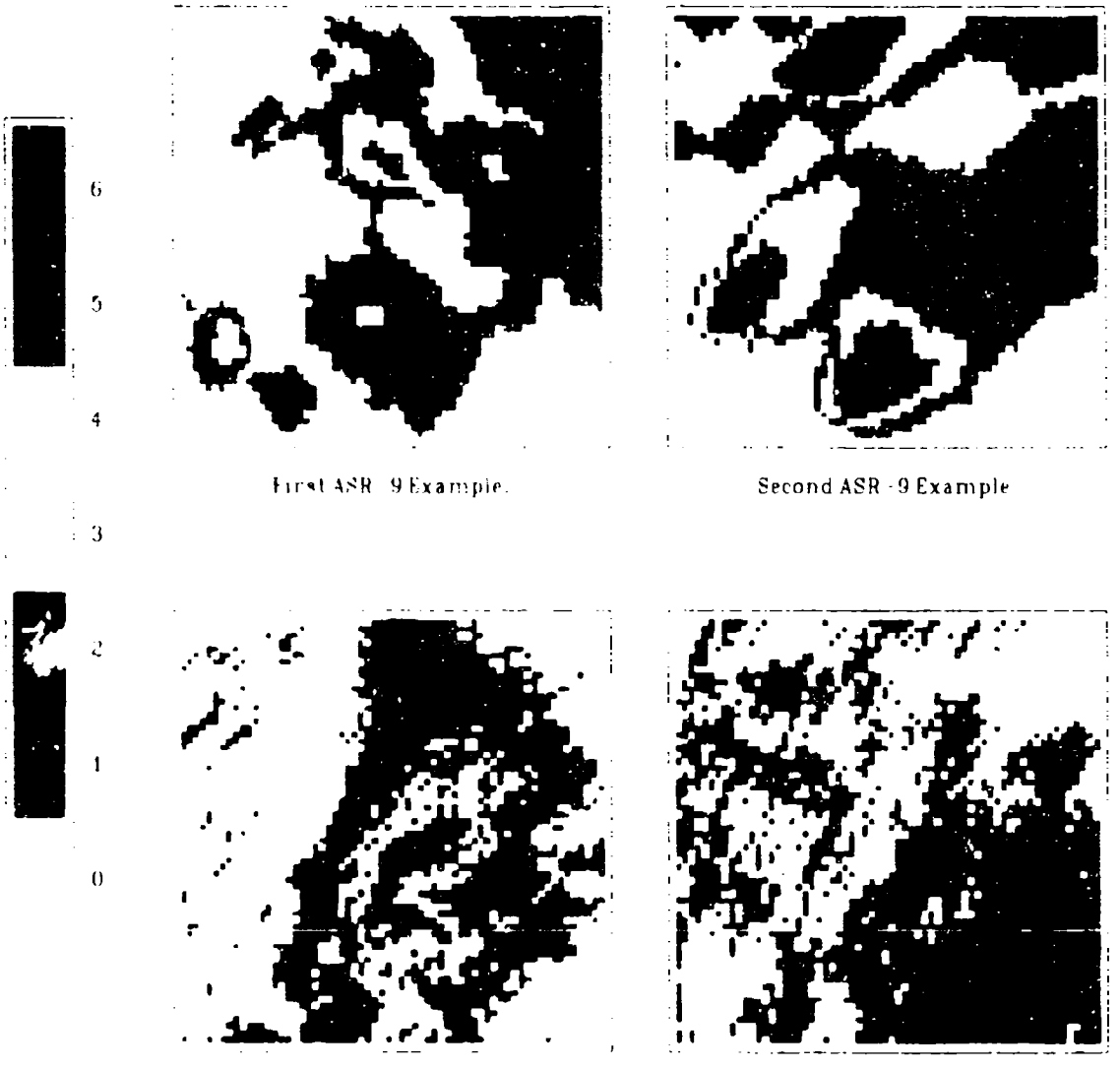


Figure 1-1 Example Weather Map Images

Table 1. National Weather Service Precipitation Levels

(vertical spacing indicates relative db range)

NWS Level	Precip. Intensity	Rainfall in/hr	Possible Turbulence	Hail	Lightning
LEVEL 6	Extreme	≥ 7.1	Severe	Large	Yes
LEVEL 5	Intense	4.5-7.1	Severe	Likely	Yes
LEVEL 4	Very Strong	2.2-4.5	Severe	--	Yes
LEVEL 3	Strong	1.1-2.2	Severe	--	Yes
LEVEL 2	Moderate	0.2-1.1	Light/ Moderate	--	No
LEVEL 1	Weak	< 0.2	Light/ Moderate	--	No

To make transmission of weather maps using data links practical, a means must be found to severely compress the image. For example, Mode S would require a 10-fold compression. In addition, the algorithms used to perform the compression and decompression must be computationally efficient: the ground computer may have to handle many aircraft requests in a given scan, and the airborne computer may neither be particularly powerful nor have extensive memory capacity.

1.3 STANDARD COMPRESSION APPROACHES

There are many techniques of data compression that are well-developed and documented. These techniques include runlength encoding, select encoding, Huffman encoding, Lempel-Ziv encoding, and arithmetic encoding [2]. These algorithms are drawn from a variety of applications including text-file archiving, compression of Fax images, and transmission of general pictures. All of these approaches make use of the redundancies in the data to achieve compression. They are general techniques, independent of the nature of the data. They also maintain exact fidelity under compression — the decompressed image exactly matches the raw image. However, as a result of this property, the approaches cannot guarantee a minimum bit limit.

Several of these approaches have been tested on sample weather-map images taken from operating weather radars. The best of these approaches, the Huffman [3], could only produce compressions of about 3:1 or 4:1 on the sample maps. None ever achieved the compression required to fit a weather-map image into a 1280-bit ELM. It became clear that an algorithm based on the unique attributes of a weather map would be required to achieve the necessary compression.

Exact fidelity under compression could not be achieved for this application; some amount of distortion in the transmission of the weather map over the data link would have to be tolerated. Therefore, the algorithms described in the remainder of this report trade off controlled amounts of distortion for increased compression. The ultimate test of these algorithms is that they produce useful maps when bit limited to 1 ELM and smaller limits.

1.4 WEATHER-MAP COMPRESSION ALGORITHMS

Weather map images exhibit a great deal of structure — they are far from random sets of pixels. If the compression algorithms can make use of this structure, the result can be the achievement of greater compression than would be possible for a "general" algorithm which simply depended on the basic redundancy of the map.

One useful property of weather-map images is their "contiguity." Weather regions tend to form smooth, continuous areas instead of isolated random spots, as seen in Figure 1-2. Abrupt edges or disconnected segments of weather are rare.

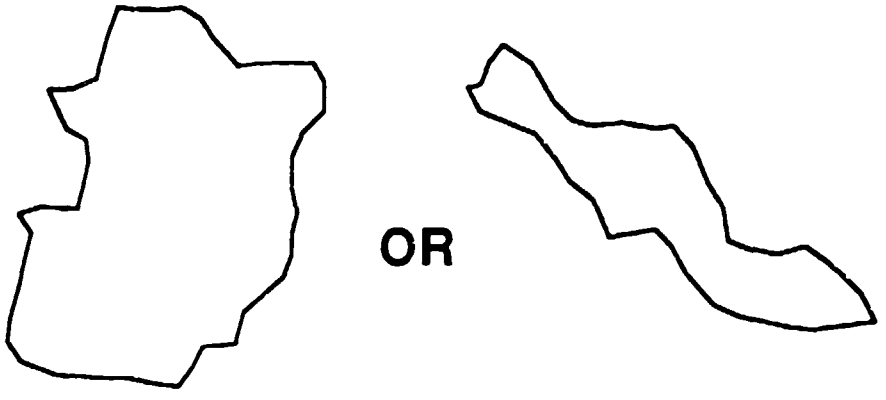


Figure 1-2. Sample Weather Regions.

A second property of weather maps might be termed "continuity" — regions of intense (higher level) weather usually are wholly contained within regions of less intense weather. In fact, weather transitions are usually restricted to plus or minus one level (see Figure 1-3.)

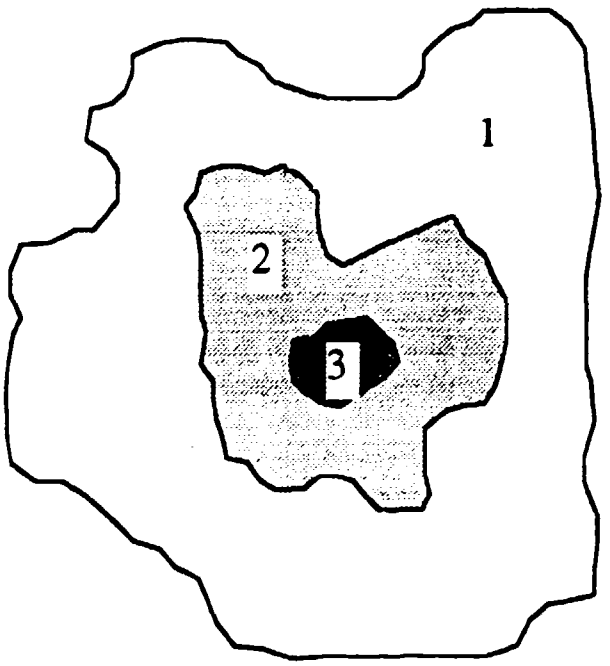


Figure 1-3. Weather Transitions.

These two properties are exploited in the two compression techniques developed in this project: The Weather-Huffman (WH) Algorithm and the Polygon-Ellipse (PE) Algorithm. The WH algorithm is a Huffman-type runlength encoding scheme that has built-in assumptions tied to typical weather map pictures; when these assumptions hold, the bit requirements of WH are far less than the theoretically optimum Huffman code. The PE approach, on the other hand, considers a weather map to be a set of geometric regions and encodes each such region with a polygon or ellipse. Both algorithms contain procedures for gracefully degrading the fidelity of the transmitted image when necessary to meet a specified bit limit. Final choice of the preferable approach awaits operational evaluations.

This report presents the full specifications of the Polygon-Ellipse algorithm. A companion report provides similar details for the Weather Huffman approach [4]

1.5 REPORT OUTLINE

Section 2 provides an executive overview of the Polygon-Ellipse (PE) approach to weather map compression. This section also includes results of applying the algorithm with a 1-ELM bit limitation requirement to sample complex ASR-9 and TDWR weather images. Finally, it estimates the ground computer and onboard avionics requirements for implementing the PE encoding and decoding routines.

Section 3 provides greater details of the approach used by the PE algorithm to trade map representation fidelity for reduction in bit requirements. It also provides an overview of the encoded data stream to be sent on the Mode S data link.

The full details of the various routines that encompass the PE approach are then provided by the remaining Sections 4 through 8. The level of detail provided in these sections is sufficient for a programmer to understand the C-coded subroutines developed as part of this project.

Finally, the Appendix lists the set of user-specified parameters that currently exist in the PE software. For each, an explanation is provided of the parameter's effect and its default setting.

2. POLYGON-ELLIPSE ALGORITHM OVERVIEW

The Polygon-Ellipse (PE) encoding algorithm represents weather regions as ellipses, polygons, and exact patterns. The actual ellipse and polygon parameters are encoded and transmitted; the decoder algorithm redraws the shapes from their encoded parameter values and fills in the included weather pixels. The PE algorithm achieves high levels of compression because most weather regions are roughly elliptical or polygonal, and it generally takes fewer bits to specify the shape parameters of the region than to specify each of the pixels making up the region. In addition, special coding techniques are used in PE to compress the encoding of the shape parameters to achieve further overall compression.

Each weather level of the map is encoded separately, and the transmitted message consists of the sequence of level encodings. Whenever bit limitations require resolution reduction for the map, some or all of the levels will be reduced from their input 64x64 resolution to their 32x32 versions prior to the encoding. The priority ordering of the levels for the reduction process can be bottom-up (level 1 reduced first) or top-down (level 6 reduced first) according to application preference.

2.1 REGION DEFINITION

The first step of the PE algorithm is the isolation of each weather region of the level L under consideration. A weather region is defined as a set of connected pixels all of which are $\geq$ L. The higher -level pixels enclosed by the level L weather are included in the region to create closed regions; these pixels will be overwritten with the correct higher level when that level is encoded.

A minimum region area threshold is defined for each weather level to permit the filtering out of small spots that would needlessly add to the encoding load. The parameters (area, moment of inertia, enclosing box) of each surviving region are then computed. Based on these parameters, the most appropriate shape — exact, ellipse, or polygon — is selected as the encoding medium.

2.2 EXACT ENCODING

Small weather regions are encoded exactly, with a bit for each pixel:

1: pixel of level $\geq$ L

0: pixel of lower level (or clear)

The parameters for the exact shape representation are the locations of the upper-left and lower-right corners of the region. This method of representation is used whenever a calculation determines that it is more efficient than the ellipse or polygon alternatives.

2.3 ELLIPSE ENCODING

Weather regions which are too large to be exactly encoded are considered for ellipse representation. A measure of the weather region's "ellipseticity" is made using the moments of inertia of the pixels making up the region. If the region's shape is sufficiently elliptical (within a parameter), it is encoded as an ellipse. The pixel locations of the two foci of the best matching ellipse and the distance parameter of that ellipse are calculated for transmission.

In order to encode the ellipse distance as an integer, yet maintain sufficient accuracy in the rendition of the ellipse, the distance is encoded in 1/4 pixel units. Therefore, for a 64x64 pixel weather map, an ellipse encoding should require 12 bits for each focus location plus 9 bits for the distance parameter, yielding a nominal total of 33 bits per ellipse. However, special encoding "tricks" generally reduce this requirement substantially.

Recreating the ellipse in the decoder uses the standard fill method, namely filling in all pixels which satisfy the relation that the sum of the distances from the pixel to the two foci is less than or equal to the ellipse distance parameter:

$$d(\text{pixel}\rightarrow a) + d(\text{pixel}\rightarrow b) \leq d$$

The PE ellipse decoder algorithm uses several techniques to speed up this operation.

2.4 POLYGON ENCODING

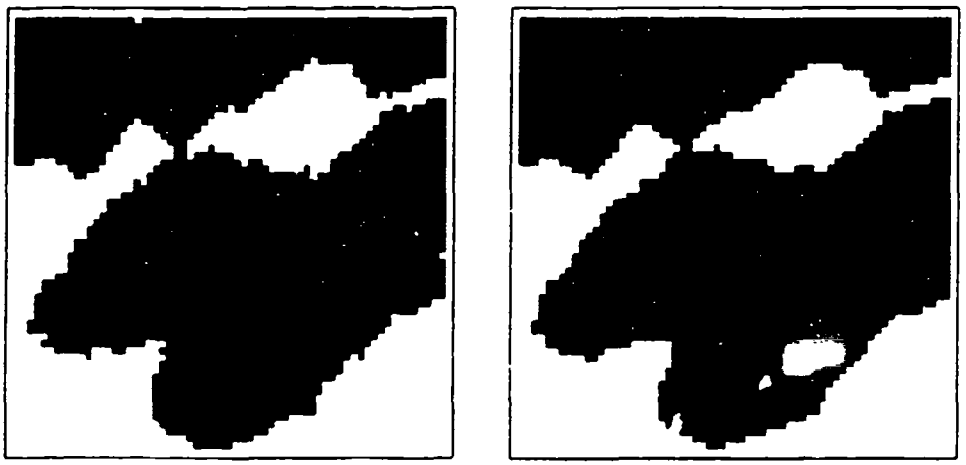
If a weather region is not sufficiently elliptical, it is fit with a polygon. The encoding is simply an ordered list of the polygon vertices. A 64x64 pixel map nominally requires 12 bits per vertex point (plus a vertex count). Again, special techniques often significantly reduce this requirement.

The first step of the polygon fitter is a tracing procedure that finds each pixel on the perimeter of the region. The second step approximates this perimeter with line segments, with segment breaks determined by the changes in slope of the perimeter. Finally, the last step is a reduction in the number of vertices until a parametric level of region distortion is reached (this setting is discussed below).

The perimeter tracing process has one significant problem: holes in the region, which are areas of low-level weather entirely surrounded by the higher level weather corresponding to the region, will be lost. When such holes are judged of significant importance, an auxiliary hole-fixing algorithm is applied before the tracing. This algorithm identifies the location of each hole, and "drills" a tunnel from the hole through the encompassing region. This transforms the hole from inside to outside the weather region, and thus the perimeter tracing leaves it intact. (The tunnel is noticed and covered by the decoder so that it is unseen by the user).

The polygon-vertex cutting procedure ("polycut") searches the vertex list to find the point that would cause the least amount of error in polygon area if it were omitted, or if it and the subsequent vertex were combined into one. Polycut weights overfill errors less than underfill errors based on the judgment that adding a weather area is less dangerous than deleting an area. Polycut continues to reduce the vertex set until the maximum permissible amount of area has been modified by the vertex cutting.

Figure 2-1 illustrates the effect on the shape of a weather region as polycut reduces the number of vertices utilized for its representation. The top picture is the true weather region. The second picture is the initial polygon fit, containing 48 vertices. Note that if bit reduction is not required, this fit is almost a perfect representation of the region. The third and fourth pictures illustrate the effects of two levels of polycut reduction: the former at 4% area modification, yielding 22 vertices; the latter at 10% area modification, yielding 14 vertices.



Input Weather Region.

Best Polygon Fit – 48 Vertices.

4% Area Modification – 22 Vertices.

10% Area Modification – 14 Vertices.

Figure 2–1. Effect on Polygon of Vertex Reduction.

The algorithm for decoding polygons simply draws the polygon perimeter lines from vertex to vertex, and then fills in the inside pixels. Although this seems simple, it is complicated by the types of polygons that can result from weather images and from the polycut procedure. Polygons that have lines sticking out, crossovers, and other unusual cases must be considered. As a result, the generalized polygon-fill procedure involves quite a bit of computation. To speed the process, the polygon encoder performs tests which determine whether a much simpler fill procedure may be used. This information is encoded as part of the polygon header.

2.5 BIT-REDUCTION PROCEDURE

The Polygon-Ellipse approach first attempts to represent the map by its best approximation to exact truth, defined as follows:

(a) all levels are set to 64x64 resolution,

(b) the "ellipseticity" parameter is set to 0.98, so that only true ellipses are so encoded, and

(c) the polygon area modification parameter is set to 0, so that no vertex cutting is permitted.

If the number of bits produced by this attempt exceeds the prescribed limit, fidelity is stepwise reduced. The first several steps lower the "ellipseticity" requirement of a level while in concert raising its polygon-modification limit. The levels are modified in cyclic order, from lowest to highest priority in each cycle. After all legal parameter changing cycles are exhausted, the levels are one by one converted to 32x32 resolution. Finally, if even this attempt fails to satisfy the bit limit (never yet encountered), levels would be represented by their single enclosing quadrilateral.

2.6 SAMPLE MAP RESULTS

Typical weather maps, representing light weather conditions, require no compression to meet a 1-ELM (1280 bits) limit. Thus, to appreciate the full representational ability of the polygon-ellipse algorithm, Figure 2-2 illustrates its performance on maps that do require compression, namely severe weather maps (the worst we could find) produced at Denver, Colorado.

The first two maps were generated from data gathered by the ASR-9 radar, a Doppler surveillance radar that also has a weather channel. This radar contains internal filtering algorithms in its weather processing, thereby producing the smoothed appearance of the weather regions. A pixel on the ASR-9 maps represents 1 square kilometer.

The last two maps were generated from data gathered by the TDWR (Terminal Doppler Weather Radar), a specialized radar built for weather detection, which has higher resolution than the ASR-9. This radar has no internal filtering, and thus produces the very granular weather appearance evident in the figures. Each pixel in these maps represents 0.25x0.25 kilometers.

The upper left map in each set shows the weather-map image input to the PE compression algorithm. The upper right map in each set shows how faithfully the PE algorithm reproduces the input map when no bit limitation is applied; the actual number of bits required by this representation is also indicated. Finally, the lower maps in each set show the results of the PE algorithm compression when a 1-ELM bit limit is enforced; the left map assumes that the highest-level weather has highest-representation priority, while the right map assumes the lowest level requires greatest fidelity of representation.

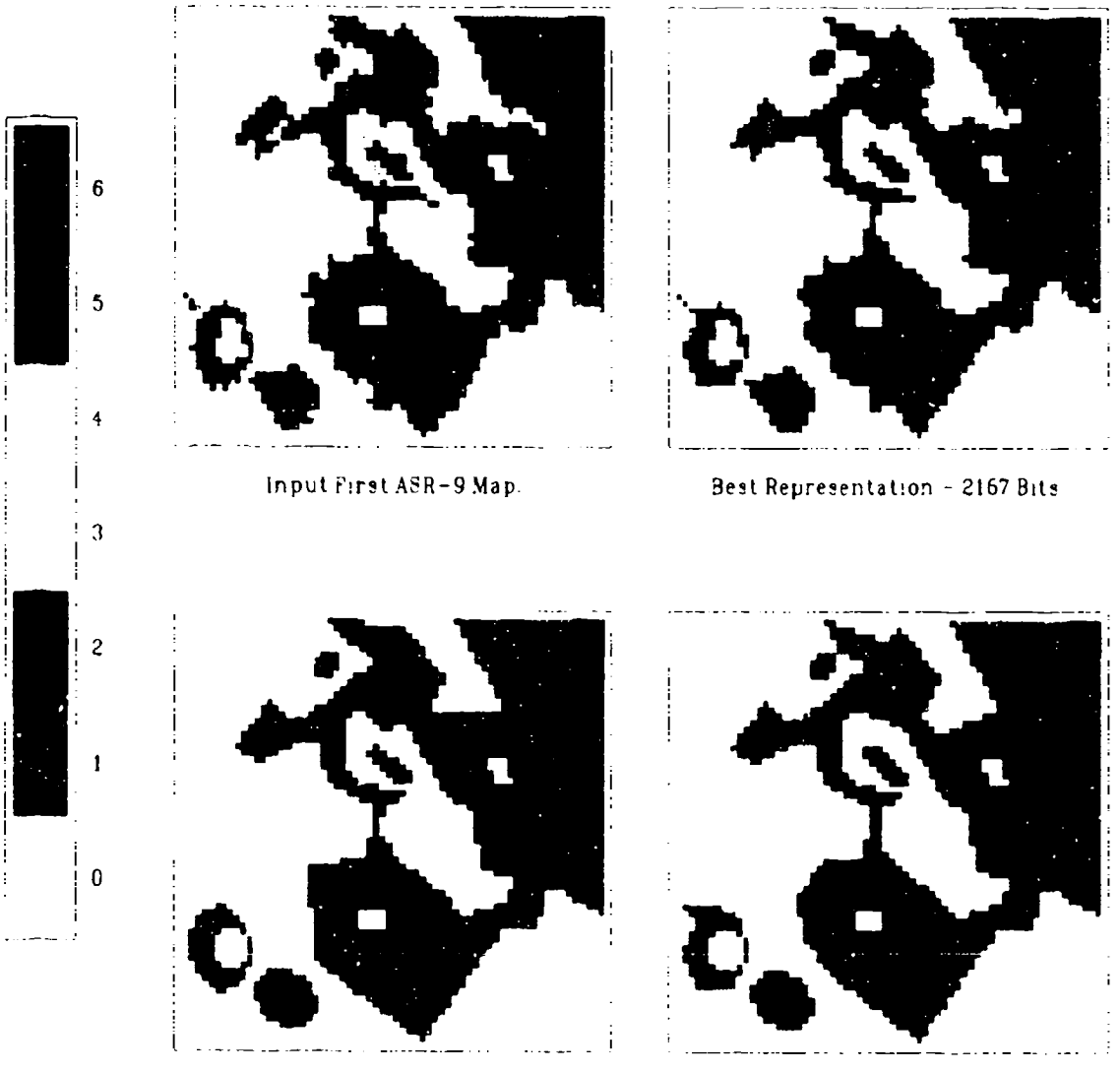


Figure 2 2a. Results of Applying the Polygon Ellipse Compression Algorithm.

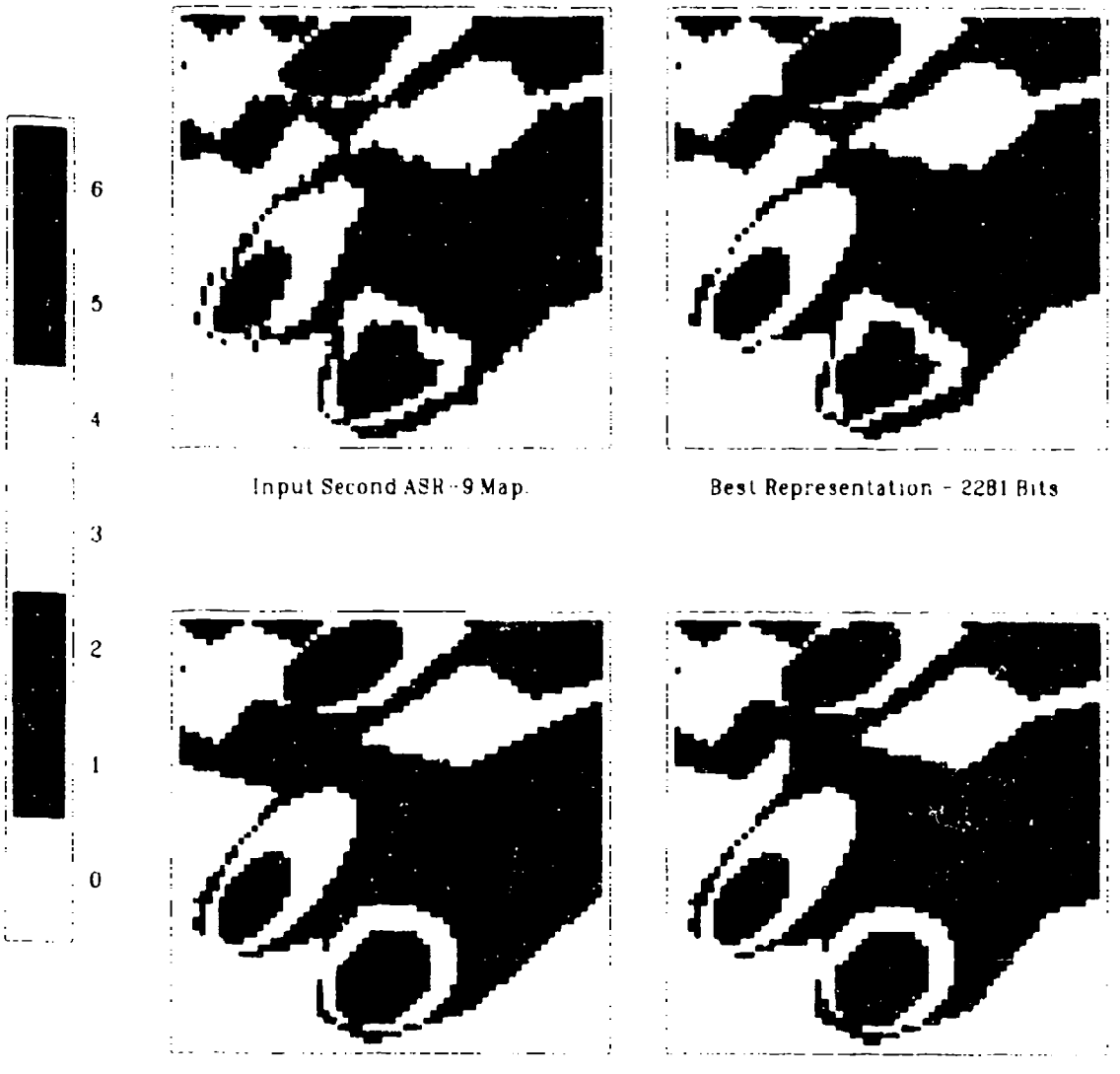


Figure 2-26. Results of Applying the Polygon-Ellipse Compression Algorithm.

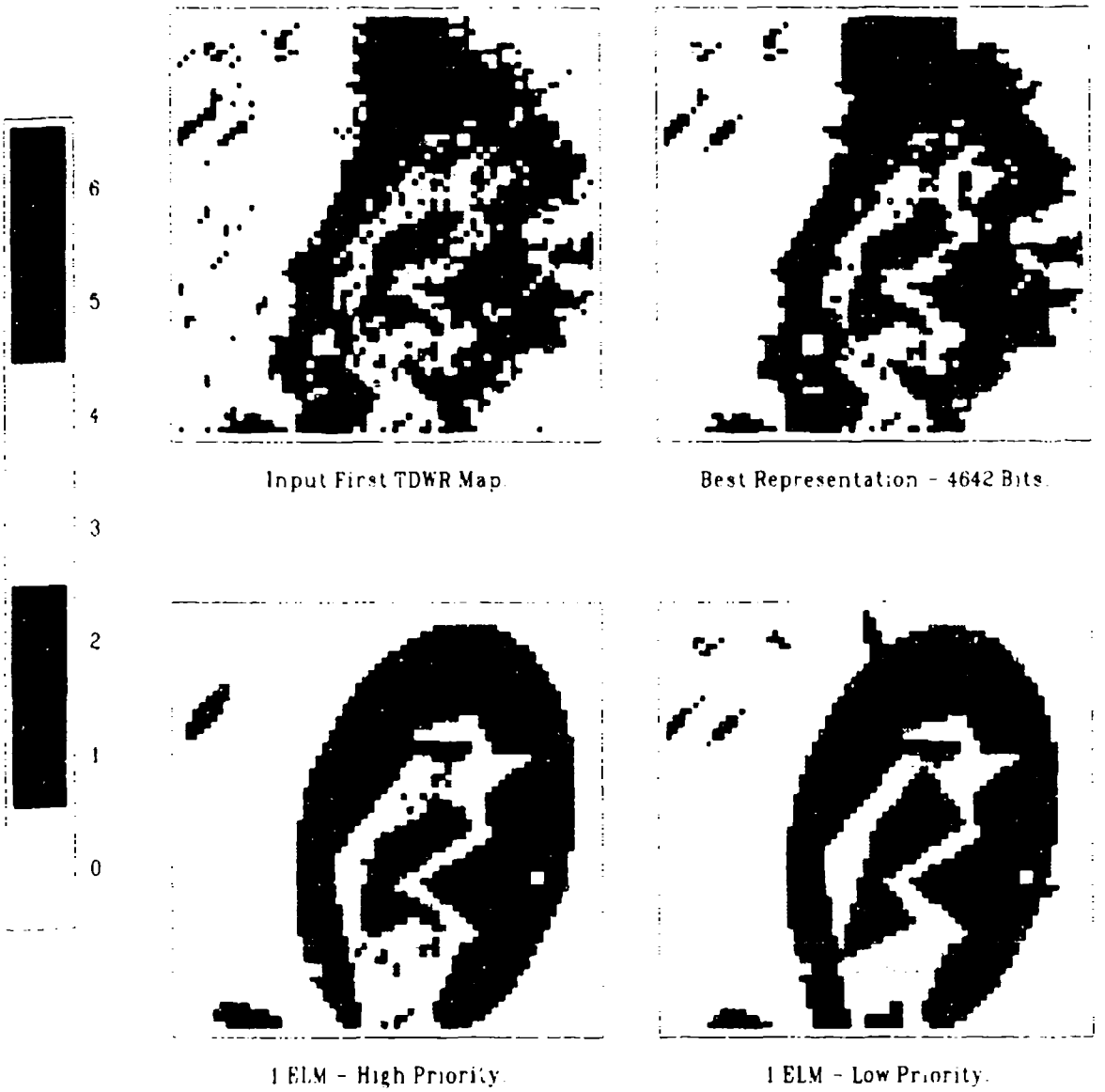


Figure 2 - 2c. Results of Applying the Polygon - Ellipse Compression Algorithm.

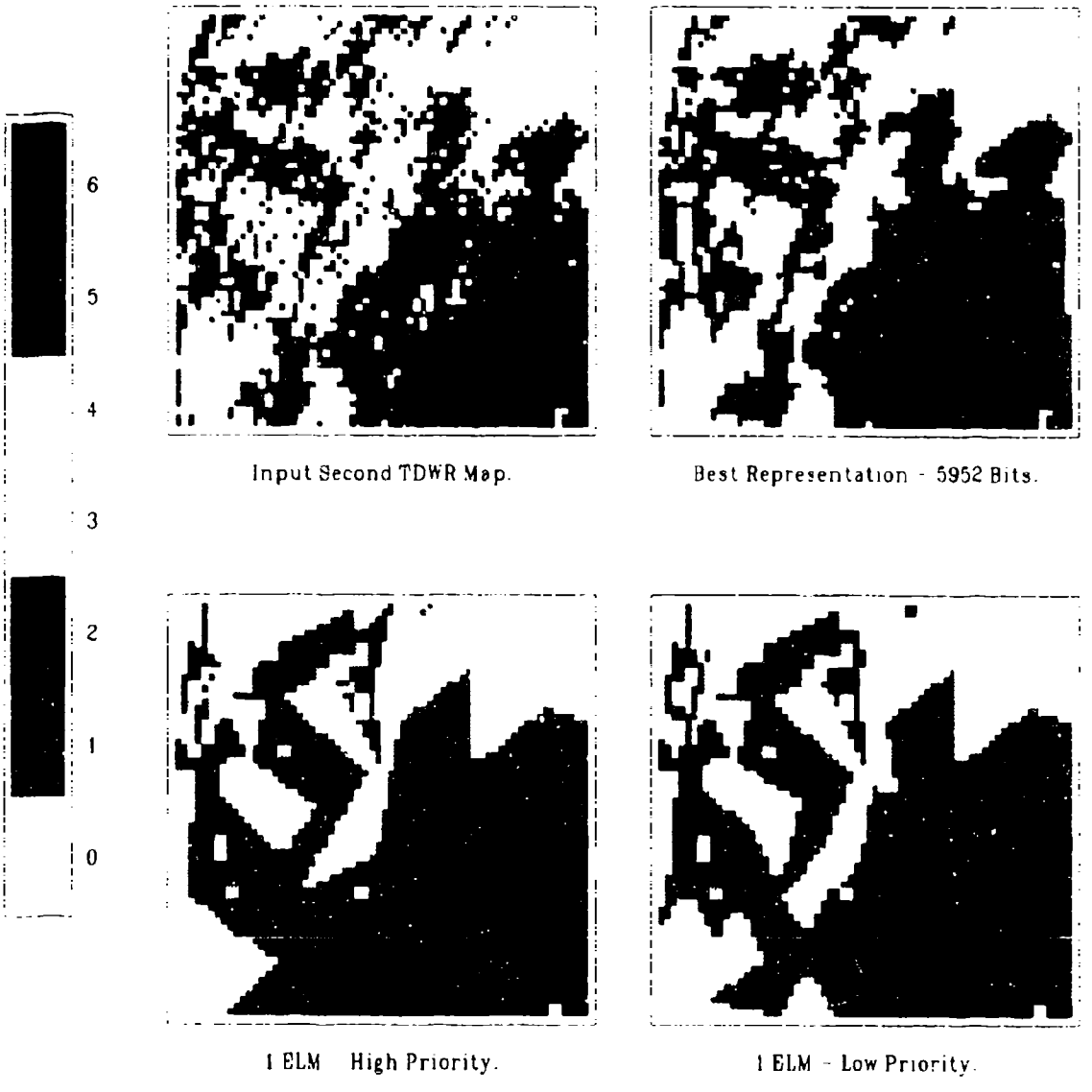


Figure 2 2d. Results of Applying the Polygon Ellipse Compression Algorithm.

It is apparent that the PE algorithm is capable of near-perfect representation of weather maps if no bit limitation is mandated. The only noticeable difference compared to the input maps is the elimination of weather "spots" from the TDWR maps. This results from the minimum weather region sizes inherent in the default parameter settings. When the minimums were all set to 1 to retain the spots, the bit requirements increased by 15% for the TDWR maps.

The tradeoffs between high-level and low-level fidelity seen by comparing the bottom two maps in each set are clear, but not as striking as one might expect. That is because high-weather-level regions tend to be few and small; thus the number of additional bits available to improve low-level fidelity by smoothing high levels is often not significant. Which priority scheme is "better" is left for aviation experts to decide.

2.7 PROGRAM SIZE AND TIMING MEASURES

The PE algorithm has been fully coded in the C language and tested on a Digital Equipment Corporation Micro VAX 3500 Color Graphics Workstation. (The 3500 is a 3-MIPS minicomputer whose performance can be matched by modern high-capability 32-bit microcomputers). Considerable effort has been made to optimize the coding of both the encoding (ground sensor-resident) and decoding (onboard-resident) routines for computational efficiency. The encoding (compression) routines start with a 64x64-pixel map-image array (read from a disk file) and produce a bit-string in memory. The decoding (decompression) routines perform the inverse transformation.

The PE algorithm requires almost 5.5 megabytes of memory to execute its compression procedure and about 50 kilobytes to execute its decompression procedure. This includes all code, data areas, and C libraries used. The large amount of memory used for PE compression is mostly composed of the precomputed data tables and save-areas used to increase performance; this requirement could be reduced by a factor of ten if speed were not critical.

The PE algorithm was tested on a set of 12 severe weather maps derived from data supplied by the ASR-9 and TDWR radars. Timing of the procedures was done with the VAX system clock, accurate to a 10-millisecond quantization. All testing assumed a 1-ELM (1280 bit) limit. The PE algorithm, as presently coded and run, required approximately 0.5 seconds on average to encode these severe maps (tests on more typical weather images produced results only one-tenth as long). The decoding routines, on the other hand, required, at most, 0.03 seconds for any complexity of map.

It is clear that the processing requirements for the airborne data link computer to perform decoding for the PE algorithm are quite reasonable:

onboard memory: 50 kilobytes

onboard processing: 0.03 seconds

The speed-optimized PE algorithm requires more computer resources to do its encoding procedure, but its requirements are still reasonable for modern ground-based computer systems.

3. POLYGON-ELLIPSE CONTROL ALGORITHM

The PE control logic is designed to incrementally degrade the fidelity of the input-weather map representation until the number of bits required to encode the resulting picture first meets the data-link limit. Since each weather level is encoded separately in the PE approach, the degradation steps have been designed to affect only one level at a time. The sequence of steps proceeds cyclically through the levels in order to maintain a semblance of equality for the amount of distortion for the levels, with each cycle proceeding from the lowest-priority level to the highest-priority one.

Two diametrically opposed priority schemes are supported by the PE software. The first assumes that since higher-weather levels are more dangerous to aircraft, they should be represented most faithfully. With this "normal" scheme, the highest weather level is given the highest-priority assignment, and level 1 the lowest priority. Other people argue, however, that no pilot would go anywhere near severe storms, so approximations to their exact boundaries are acceptable. On the other hand, pilots will try to plot their courses around moderate rain areas, and so the lower levels should be represented as accurately as possible. With this "reverse" scheme, level 1 is assigned the highest priority and the highest-weather level, the lowest priority.

3.1 LEVEL BIT-REDUCTION PASSES

At each successive bit-reduction step, the control logic establishes a new ensemble of values for the critical distortion parameters. For each level, these parameters are:

(a) level resolution

The resolution can be the full input 64x64-pixels, or be reduced to 32x32 by a 2x2-into-1 pixel mapping.

(b) "ellipseticity" factor, (i.e., 2x2-into-1 pixel)

This factor specifies the least ellipse-like a weather region can be and still be encoded with an ellipse representation.

(c) polygon area modification factor

This factor specifies the upper percentage bound to the weather area modification that is allowed as a result of reducing the number of polygon vertices.

Because of the cyclic nature of the steps, the parameters for a given level change only when that level's turn to degrade occurs. Each different set of parameters for a level will constitute a new pass through the encoding logic for that level.

As an example of successively increased distortion through parameter modifications, the complete set of passes, and their assigned distortion parameters, used for the lowest priority level are shown in Table 2.

Table 2. Distortion Settings versus Processing Pass, Lowest Priority Level.

Pass	Resolution	Ellipse Factor	Polygon Area Mod.
0	64x64	.98	0 %
1	64x64	.98	2 %
2	64x64	.97	4 %
•	•	•	•
•	•	•	•
•	•	•	•
10	64x64	.93	20 %
11	32x32	.92	20 %
12	32x32	.91	25 %
13	32x32	.90	30 %
14	32x32	0 (ellipse forced)	—
15	32x32	—	4-point polygon for whole level

Higher priority levels, as explained in Table 3, require fewer passes, as they are distorted less often than lower priority levels. In fact, each subsequent level has one fewer pass in the 64x64 section. In addition, since the parameters in the 32x32 section of the chart are incremental variations from the lowest 64x64 pass, less distortion exists in this section's passes for each higher priority level. Thus, in particular, the chart for the highest priority level (priority 6) has five fewer passes.

Table 3. Distortion Settings versus Processing Pass, Highest Priority Level.

Pass	Resolution	Ellipse Factor	Polygon Area Mod.
0	64x64	.98	0 %
1	64x64	.98	2 %
2	64x64	.97	4 %
•	•	•	•
•	•	•	•
•	•	•	•
5	64x64	.96	10 %
6	32x32	.95	10 %
7	32x32	.94	15 %
8	32x32	.93	20 %
9	32x32	0 (ellipse forced)	—
10	32x32	—	4-point polygon for whole level

Note that the passes can be divided into four categories:

1. Full resolution, ellipses and polygons employed to represent weather regions
2. Reduced resolution, ellipses and polygons employed to represent weather regions.
3. Reduced resolution, only ellipses employed to represent weather regions.
4. Reduced resolution, one quadrilateral used to represent the union of all weather regions

Each of the first two categories consist of several passes, with the change from one to the next being that more regions become eligible for representation by ellipses (ellipses require fewer bits to specify than polygons, but are often less faithful to region shape), and more distortion is permitted in polygon representations (thus fewer polygon vertices are needed).

Figure 3-2 illustrates the reduction in representation fidelity of a weather map as the pass numbers utilized for the levels increase; level 1 highest priority is assumed for this example. The upper-left map is the result produced when all levels are at their initial pass 0. The upper-right map results when all levels have progressed to the final pass of the first category. As seen, several of the polygons have converted to ellipses, and the remaining polygons have fewer vertices. The lower-left map then shows the result when level 1 is at the last pass in category 2 while all other levels are at category 3, ellipses only. Although level 1 still employs polygons, the fidelity has degraded significantly from the previous map. Finally, the last map results when levels 1 through 3 are forced to ellipses and the higher levels are forced to quadrilaterals. Note, as described above, that a single quadrilateral is used to represent all three level 4 regions.

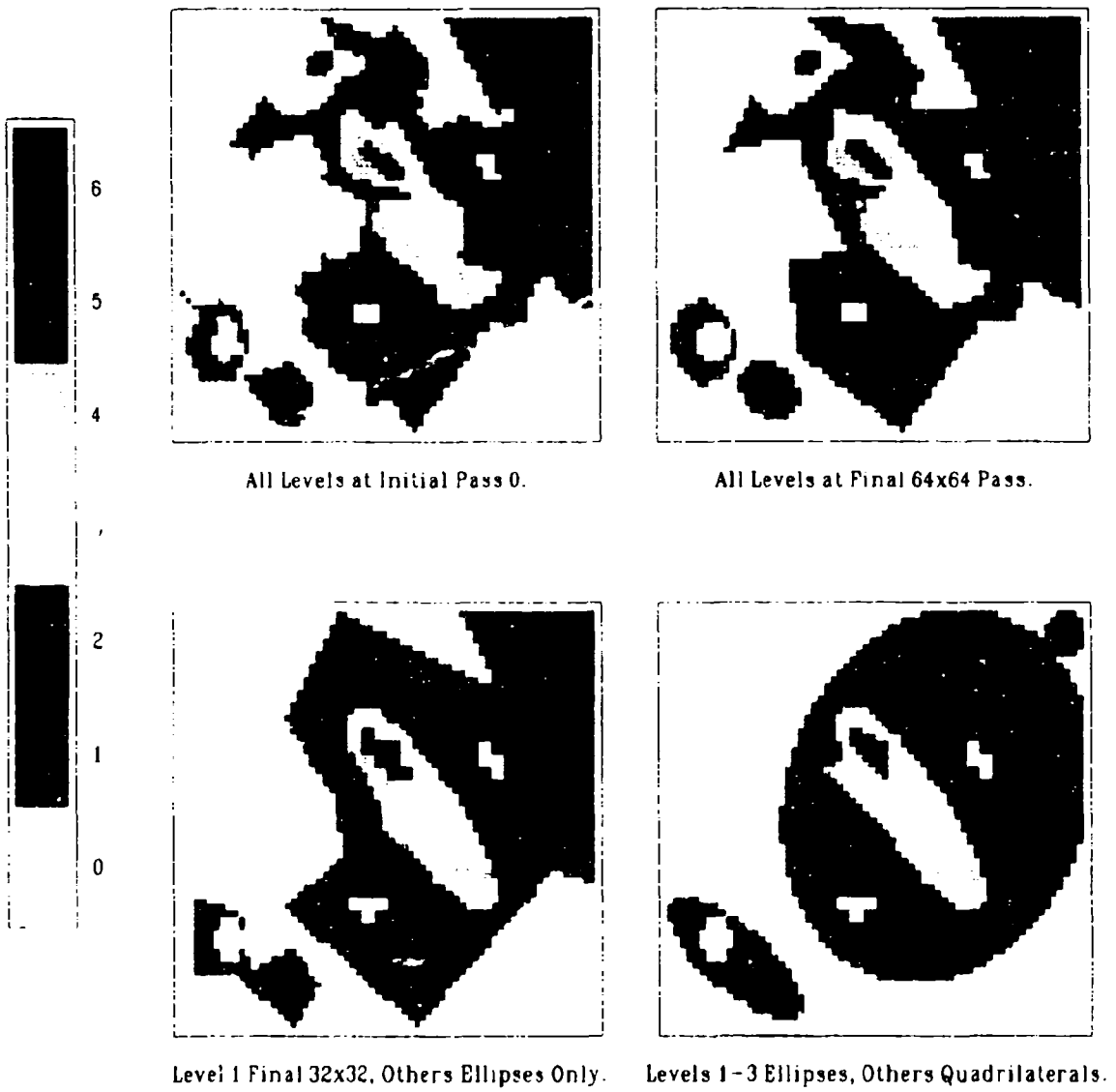


Figure 3-2. Fidelity Reduction with Pass Number Increase.

3.2 CONTROL-LOGIC INCREMENTAL STEPS

The rule that determines which level is next to undergo further distortion, when the bit limit is still not satisfied, is that a lower priority level should always have more distortion than a higher priority level. The algorithm that embodies this apparently simple rule is the following, where, for simplicity, level 1 is assumed to be the lowest priority level:

1. Initialize all levels to pass 0.
2. Set the initial distortion parameters for each level.
3. Encode all levels of the map.
4. If the map encoding bit limit is met, stop.
5. Otherwise, find the largest level L for which Pass(L) $\leq$ {Pass(L-1) - 2}. If none, set L = 1.
6. Increment the pass number for level L.
7. Recode level L using its new distortion parameters.
8. Return to step 4.

In words, rule 5 says that a high priority level has its distortion increased only when, by so doing, its distortion still remains less than that of the next lower priority level. The actual sequence of pass-to-level assignments resulting from this algorithm is as shown by Table 4.

In actual implementation, the encoding time implied by this algorithm is significantly reduced by a simple modification. Namely, map level L is not encoded until the number of bits required to encode levels above L in priority satisfies the bit limit. As soon as the top-down encoding exceeds the limit, the next sequence number applied is the first one that increases the set of passes for the levels already encoded. For example, if the initial sequence 0 encoding fails at level 3, sequence number 6 is utilized next, as this number is the first that increases level-3 distortion. Then, if this recoding of level 3 removes the bit-limit violation, level 2 is encoded at pass 2, skipping the need for prior codings at passes 0 or 1. For very complex maps, it is often found that some of the lower levels never get encoded at 64x64 resolution at all.

Table 4. Sequence of Level Pass Assignments.

Sequence #	Pass(1)	Pass(2)	Pass(3)	Pass(4)	Pass(5)	Pass(6)
0	0	0	0	0	0	0
1	1	0	0	0	0	0
2	2	0	0	0	0	0
3	2	1	0	0	0	0
4	3	1	0	0	0	0
5	3	2	0	0	0	0
6	3	2	1	0	0	0
---	levels experiencing initial distortions					
21	6	5	4	3	2	1
22	7	5	4	3	2	1
---	levels all at 64x64, distortion increasing					
45	10	9	8	7	6	5
46	11	9	8	7	6	5
---	levels proceeding in order to 32x32					
51	11	10	9	8	7	6
52	12	10	9	8	7	6
---	levels all at 32x32, distortion increasing					
63	13	12	11	10	9	8
64	14	12	11	10	9	8
---	levels proceeding in order to all ellipses					
69	14	13	12	11	10	9
70	15	13	12	11	10	9
.	levels proceeding in order to 1 quadrilateral					
75	15	14	13	12	11	10

3.3 LEVEL PROCESSING

Each time the PE master control logic increases the distortion parameters for a given level, the weather regions on that level must be reprocessed, and new shape representations and encodings generated. In general, exact specifications, ellipses, and polygons are the available ensemble for the weather regions. The one exception occurs if the number of bits is so severely limited that the entire level must be represented by a single quadrilateral.

The first task to be performed on a single-level map, either the original 64x64 map or the reduced-resolution 32x32 map is the identification of the individual weather regions. This is performed by a labeling operation on the level's non-zero pixels; the details of the labeling are presented in Section 4. Since the labeling of a region is independent of the distortion parameters to be applied to the region, it need only be performed once for each map, and the results preserved in memory.

Once the individual regions are known, each one whose size exceeds the minimum parametric value must be encoded by the appropriate type of shape. The shape selected is the legal alternative (the ellipse may be rejected if the region is not elliptical enough) that requires the fewest number of bits. The decision logic is as follows:

1. Calculate $exact_i$, the number of bits required to encode region i by exact specification.
2. Calculate the "ellipseticity" factor f_i of region i and determine the specification parameters of the ellipse that best represents the region.
3. Calculate the number of bits $poly_i$ required to encode the polygon that best represents region i at the specified area-modification distortion level.
4. If $exact_i \leq$ bpe (the estimated fixed number of bits to encode an ellipse), encode region i by exact specification.
5. Else if $f_i \geq F_{p,L}$ (the minimum ellipse factor for pass p at level L), encode the region as an ellipse.
6. Else if $exact_i \leq poly_i$, encode region i by exact specification.
7. Else encode region i as a polygon.

Step 5 is the one that checks for the legality of using an ellipse.

Once again, the value of $exact_i$ in step 1 and the "ellipseticity" and ellipse specification parameters of a region in step 2 are independent of distortion parameters, and thus are computed once and stored in memory for future reference. Although the polygon vertices for a region are strongly dependent upon allowable distortion, and so must be recomputed for each pass, the initial tracing of the region need only be performed once; thus the results of this process are also stored in memory. Subsequent sections of this report provide detailed explanations of the exact specification, ellipse, polygon, and single quadrilateral representation algorithms.

3.4 MAP-ENCODING PROCEDURE

For the PE map-representation approach, each weather level is encoded separately. Thus, the overall encoded message is a concatenation of individual-level encoding streams. Furthermore, since

each weather region on a level is encoded separately, the individual-level encoding stream is further subdivided into individual-region encoding representations.

The header of the overall encoded message provides two pieces of information:

1. The number of the highest weather level L_{max} that actually exists on the map.
2. The lowest priority level L_{min} that is encoded with full 64x64 resolution.

Note that since the switch from full 64x64 resolution to reduced 32x32 resolution proceeds one level at a time, from lowest priority level to highest priority level, stating the cutoff level L_{min} is sufficient; the resolution of each level does not have to be specified individually.

The header for each of the L_{max} individual levels contains one piece of information:

1. The number of weather regions R_i encoded on level i.

This number is 1 if all the regions on the level end up being represented by a single quadrilateral.

The header for each of the R_i weather-region specifications on level i indicates whether the region is encoded by exact specification (square or rectangle), by an ellipse, or by a polygon (a quadrilateral is merely a special polygon). To save some header bits, all ellipses are placed at the end of the level list; when the first ellipse is encountered, no further headers are required for the level. The details of the encoding for each shape is spelled out in detail in Sections 5, 6, and 7, respectively.

3.5 MAP-DECODING PROCEDURE

The decoding procedure for the PE algorithm follows from the encoding rules just presented. In particular, the steps are as follows:

1. Read V, the value of the first 3 message bits.
2. Set the highest map level to $L_{max} = V$.
3. Read V, the value of the next 3 message bits.
4. Set the lowest priority 64x64 level to $L_{min} = V$.
5. Initialize the level to L = 1.
6. Set the number of regions on level L to n = 0.
7. Read V, the value of the next 3 message bits.
8. Increase n by V.
9. If V < 7, stop; else if V = 7, return to step 7.
10. If N = 0, proceed to step 17.
11. Else, initialize the region count to n = 1
12. Initialize the specification type to T = 10.
13. If $T \neq 0$, read V, the value of the next 2 message bits.
14. If $T \neq 0$, set T = V.

15. If T = 0, read and decode the ellipse specification,

    Else if T = 1, read and decode the polygon specification,

    Else if T = 2, read and decode the square specification,

    Else read and decode the rectangle specification.

16. Increment n by 1; if $n \leq N$, return to step 13.

17. Increment L by 1; if $L \leq L_{max}$, return to step 6, else stop.

3.6 OUTPUT WEATHER-MAP GENERATION

The output weather map that will be displayed onboard the aircraft is generated incrementally as each weather-region specification is decoded by the above procedure. Initially, the output map is set to all pixels clear of weather, that is $pixel_{i,j} = 0$. As each shape is decoded, the fill procedure for that shape is invoked. The fill algorithms for exact shapes, ellipses, and polygons are provided in Sections 5, 6, and 7, respectively.

The fill procedure for each newly decoded shape overwrites whatever values are currently contained in the filled pixels. Since the levels are decoded in order of intensity, this overwriting guarantees that each pixel will end up with its proper weather-level setting.

4. WEATHER-MAP PREPARATION

The polygon-ellipse algorithm encodes, level by level, each sufficiently large weather region on the input 64x64-pixel, 7-level weather map. In order to proceed, several pre-processing functions must be performed on the map.

1. Produce 6 individual-level 64x64-pixel weather maps, one for each of the non-zero weather levels.
2. Produce 6 individual-level 32x32-pixel weather maps to be used if bit limitations force a resolution reduction for some or all of the weather levels.
3. For each individual-level map, modify weather regions as necessary to preserve weather holes.
4. For each individual-level map, identify and label each separate weather region.

The PE algorithm performs all its encoding logic on these individual-level maps.

4.1 CONSTRUCTION OF 64x64-PIXEL SINGLE-LEVEL MAPS

Each single-level map is a 0-1 weather map, with 1 indicating a weather pixel to be encoded and 0 representing no weather. In each case, the 0 or 1 refers only to the point of view of the specific level being represented by the map; the original input map could indicate lower level weather present at a 0 of a single-level map.

In general, weather tends to be nested, with more severe levels contained within lower levels (see Figure 4-1).

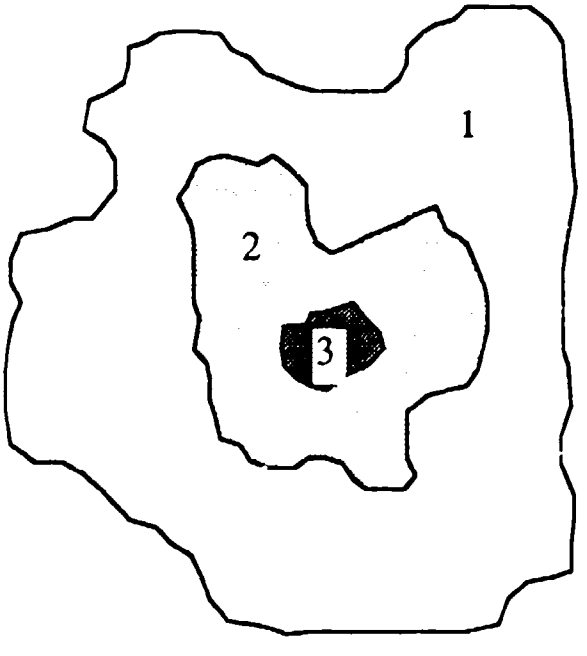


Figure 4-1. Typical Nesting of Weather Levels.

Thus, if the level 1 single-level map did not include higher weather levels as counted points, the weather regions would tend to be toruses. Attempting to represent a torus by an ellipse of equal area would produce a very poor match; the middle hole would be covered by the ellipse, while the actual level 1 weather would hardly be represented at all.

For this reason, a "1" on the single-level map is used to represent any pixel having weather at or above the level of the map. Fortunately, adding the higher level weather to the single-level map cannot cause any misrepresentation after final message decoding. The shapes produced when higher level single-level maps are encoded will overwrite the lower level ones at the pixels where the higher level weather exists.

Thus, the procedure for defining a single-level map of level L is to scan the input map M and set the pixels of the new map according to:

$$L_{ij} = \begin{cases} 1 \\ 0 \end{cases}$$

All single-level maps can obviously be produced during a single pass through the original map; if $M_{ij} = P$, L_{ij} is set to 1 for all levels $L \geq P$.

Also during this process, for each single-level map, the minimum and maximum non-zero pixels for each row are recorded. These variables, $min_{L,i}$ and $max_{L,i}$ respectively, come in very handy for the later region labeling operations. Note that because of the nesting definition of the single-level maps:

$$min_{L,i} \leq min_{P,i} \quad P \geq L$$

$$max_{L,i} \geq max_{P,i} \quad P \geq L$$

This permits an optimized approach to setting the row minimums and maximums during the single input map scan. Namely, if $M_{ij} = P$, only the level P values need be adjusted, rather than all levels P and above as might be expected. Then, after the scan is complete, the final values are set top-down by level as:

$$min_{L,i} = \underset{P \geq L}{Min} \{ min_{P,i} \}$$

$$max_{L,i} = \underset{P \geq L}{Max} \{ max_{P,i} \}$$

4.2 CONSTRUCTION OF 32x32-PIXEL SINGLE-LEVEL MAPS

Should the input weather map be so complex that the encoding algorithms are incapable of meeting the message-bit limitation even after all shape simplification measures have been attempted on the 64x64 maps, some or all of the single-level maps must be reduced from 64x64 resolution to 32x32. To produce this reduction, each 2x2 region of the original input picture is mapped into a single pixel of a new 32x32 map; this map then serves as the reference for the single-level 32x32 maps.

Since high-level weather is considered too important to miss, the presence of even a single high-level pixel in the 2x2 region is sufficient to require the new pixel to be set to that value. With lower-level weather, however, a degree of averaging is permissible. To implement these design criteria, each level is assigned a pixel score according to the formula:

$$Score_L = 5^{L-1}$$

which guarantees that the identity of the highest level weather pixel in the 2x2 region will be maintained (since $Score_L > 4 \cdot Score_{L-1}$).

Then the setting of the pixel in the reduced map is determined from the sum of the scores of the pixels in the 2x2 region of the original map that it represents. The present mapping is shown in Table 5, where level 4 is defined to be the level at which the switch to high-level dominance occurs.

Table 5. Setting of 2x2 Pixel versus 2x2 Score.

2x2 Score	Pixel Level
0 - 1	0
2 - 8	1
10 - 40	2
50 - 100	3
125 - 500	4
625 - 2500	5
3125 - 12,500	6

Examples of the application of this table are as follows:

$\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$ maps to a 1 (averaging allowed at low levels)

$\begin{bmatrix} 2 & 3 \\ 4 & 3 \end{bmatrix}$ maps to a 4 (no averaging allowed at high levels)

The process of constructing the reduced-resolution, single-level maps is undertaken during the same pass through the input map as described in Section 4.1. In particular, the 2x2 region scores are built from the values of each pixel as they are read. When all 4 pixels in any 2x2 region have been scanned, the pixel value for the overall 32x32-pixel map is determined from the sum of its 4 scores. Then, the 0-1 values and the row minimums and maximums for the reduced-resolution, single-level maps are calculated in the same manner as for the single-level 64x64-pixel maps.

4.3 MAINTAINING WEATHER HOLES

On occasion, areas of light weather (or even clear areas) will exist within a ring of more severe precipitation. Such areas will be denoted weather "holes." The problem with having a hole within a weather region is that should the region be encoded by a polygon, the polygon decoding algorithm will fill in all pixels within the contour of the polygon; thus, the hole will be lost.

Any attempt to embellish the polygon algorithms to account for the presence of holes, such as by defining negative polygons to "erase" interior points, would add substantial complexity and execution time to the chosen approach, particularly to the onboard decoder, and thus must be rejected. Instead, a simple "trick" was adopted to alleviate the situation. Note that holes are a problem only when they are contained inside the polygon contour. But if a "tunnel" is drilled from

the outside world through to the hole, the hole is magically transformed to be outside the polygon (see Figure 4-2).

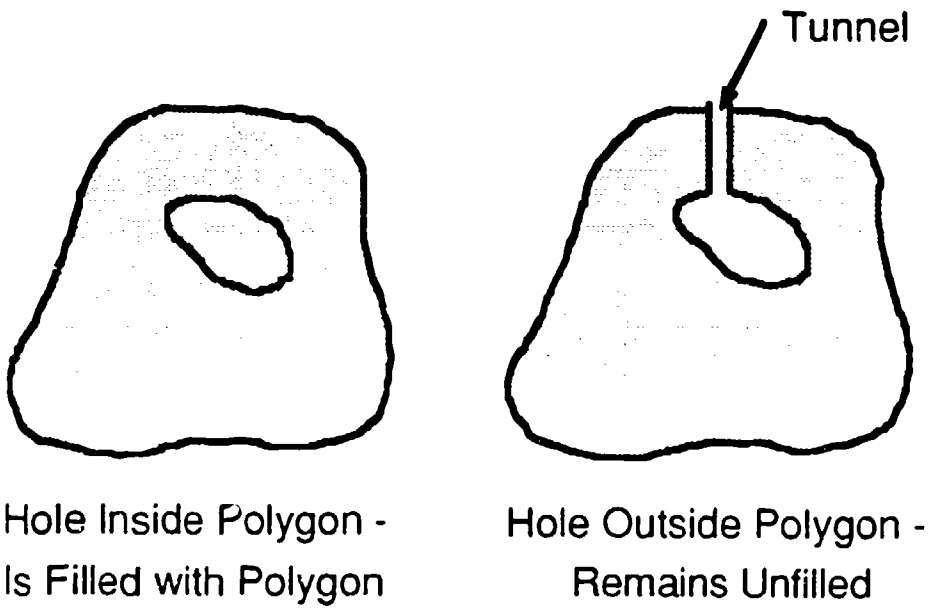


Figure 4-2. Transforming a Hole Location via a Tunnel.

Now when the region is traced, the hole will not be enclosed. Then, when the polygon produced for the region is eventually filled by the decoder, the hole will be left blank. The pixels in the tunnel will not show on the final map as they are recognized by the decoder and covered over.

In general, holes within low-level weather are of more interest to pilots than holes within severe storms. In the former case, the pilot might like to vector to the clear areas to simplify the flight. But few pilots would fly through a severe hail storm to get to a less severe thunder storm area within it! Thus the PE algorithm only seeks out holes in levels up to a user-input maximum level. Also, independent of the minimum-size weather-region parameter for a level, the smallest hole that is ever processed is 6 pixels.

The procedure for "saving the holes" has two parts: (1) identifying the regions that constitute holes, and (2) drilling the tunnel to the outside world. Since the hole identification process is a modification of the region-labeling algorithm presented in Section 4.4, further algorithm description will be tabled until Section 4.5.

4.4 WEATHER-REGION IDENTIFICATION

In order to encode a weather region, all the pixels in the region must first be identified. The procedure that accomplishes this goal is the map-labeling operation; this operation scans a single-level map row by row, labeling each non-zero pixel with the number of the region to which it

belongs. The labeling algorithm described in this section is an enhanced (and corrected) version of the algorithm presented in [5].

Each non-zero pixel A (at row i, column j) encountered during the scan is labeled as belonging to a region by referring to the previously set region labels of its three neighbors B, C, and D, defined as follows:

	column j-1	column j
row i-1	C	B
row i	D	A

If all three neighbors of A are zero, a new region is started with pixel A by assigning it the next sequential number. Otherwise, A is labeled using the label of a non-zero neighbor; if two or more are non-zero, the order of priority is to use C, then B, and finally D.

The one complication in this approach occurs when C is zero and B and D have different labels. This indicates that two previously thought disjoint regions are now seen to be connected. This would occur, for example, when the point indicated below by "?" is encountered:

```
1  1  1  0  0  2  2
1  1  1  1  0  2  2
1  1  1  1  1  ?
```

The solution to this problem is to maintain equivalence sets of region labels; when the situation $B \neq D$ is encountered, a new equivalence entry is created.

Note that this equivalence entry between B and D must be made even if the ? pixel is zero, and no labeling need be assigned to it; otherwise regions joined at a right diagonal would incorrectly be thought to be non-connected. Thus, zero pixels cannot just be skipped (contrary to [5]).

To summarize, the labeling algorithm consists of the following rules:

1. If A is zero, leave the label at 0, otherwise

   if C is non-zero, label A = C, otherwise

   if B is non-zero, label A = B, otherwise

   if D is non-zero, label A = D, otherwise

   assign the next available unused label to A.

2. If C = 0,

   if $B \neq 0$, and $D \neq 0$, and $B \neq D$,

   record an equivalence between B and D.

Note that the equivalence problem of rule 2 cannot occur if C is non-zero, as a by-product of this algorithm is that any two consecutive non-zero pixels in a row or column will always be labeled as equal or equivalent. Thus, the presence of a non-zero C guarantees B and C are labeled equivalently, as must be C and D, so that B and D are already equivalent by the transitive law of equality.

Since the first and last weather pixels in a row, min_i and max_i, are known, a few simplifications to the pixel-by-pixel scan are possible. In particular, empty rows can be skipped, and the pixels in a row following an empty row need only be compared to the D pixel.

As each weather region is labeled, its area is determined. Although the region will in general contain pixels at various weather levels, only the pixels at the level of the specific single-level map are of interest when determining whether a region's area satisfies the minimum size requirement; higher-level pixels will be re-encoded when their own level maps are considered.

After all labeling is completed, the label of each non-zero pixel is replaced by the lowest number in its sub-region's equivalence class. For example, if an equivalence set is {2,4,11}, all pixels labeled 4 or 11 are set to 2. This simplifies future processing by insuring that all pixels in a region have the same label. Also, the size of the region is determined as the sum of the sizes of all the equivalenced sub-regions.

4.5 HOLE IDENTIFICATION

Hole identification is also a labeling process, only now the 0s of the single-level map must be labeled instead of the 1s. Two other modifications to the above algorithm are required for labeling holes:

1. The map to be labeled is made 66x66-pixels, with rows and columns from 0-65. That is, the edges (all 0) are now considered part of the map, although their pixels are not processed, but are left as being labeled 0.
2. No diagonal checking is utilized.

The reason for expanding the map to include edges is to facilitate the determination of which labeled regions are "outside" and not of concern, and which are "inside" and are indeed holes. The simple rule is that:

> the region labeled 0, and all regions equivalenced to region 0, are not holes; all other regions are holes.

This rule is equivalent to saying that any region connected to an edge of the map is an outside region, and any region isolated from all edges must be inside a weather region and thus be a hole.

The elimination of diagonal checking for hole regions is required simply because the weather-region labeling algorithm utilizes it. Thus, if 0 pixels meet only at a diagonal, weather pixels must meet at the cross diagonal and be joined into a single region. For example, consider the following situation:

0	0	0	0	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	0
0	0	0	1	0	0	0	1	1	1
0	0	1	1	0	0	0	1	1	1
0	0	1	1	1	1	1	1	1	1

Clearly, the italicized 0 pixels constitute a hole of the enclosing polygon of 1s, even though the underlined pixels are connected diagonally. Thus, in summary, the complementary labeling algorithm for potential hole regions simplifies to:

	column j-1	column j
row i-1	–	**B**
row i	D	**A**

1. If A = 1 (a weather pixel), leave the label at 1, otherwise

   if B ≠ 1, label A = B, otherwise

   if D ≠ 1, label A = D, otherwise

   assign the next available unused label to A.

2. If A ≠ 1,

   if B ≠ 1, and D ≠ 1, and B ≠ D,

   record an equivalence between B and D.

Having knowledge of the first and last weather pixels in each row, namely the values min_i and max_i discussed above, allows a significant processing improvement to be made to the nominal pixel by pixel map scan for hole identification. In particular, all 0 pixels in rows i-1, i, and i+1 outside the limits for row i can automatically be left labeled as 0:

$$\text{label } p_{kj} = 0 \qquad (j < min_i) \text{ or } (j > max_i) \qquad k = i-1, i, i+1$$

This follows by definition of min_i and max_i for row i; for row i-1, all 0s not labeled as 0 would become equivalenced to 0 by rule 2; for row i+1, all 0s would be labeled 0 by rule 1 (as B=0).

Once the interior hole regions are identified, a tunnel from each to the outside world must be drilled. For simplicity, the tunnel is always begun from the known upper left pixel of the region. The tunnel is allowed to be drawn up or down; the direction chosen is the one whose length outside the hole is shorter. The tunnel can end only when a 0 pixel labeled as outside is encountered. Thus, a sample tunnel would be as seen in Figure 4-3.

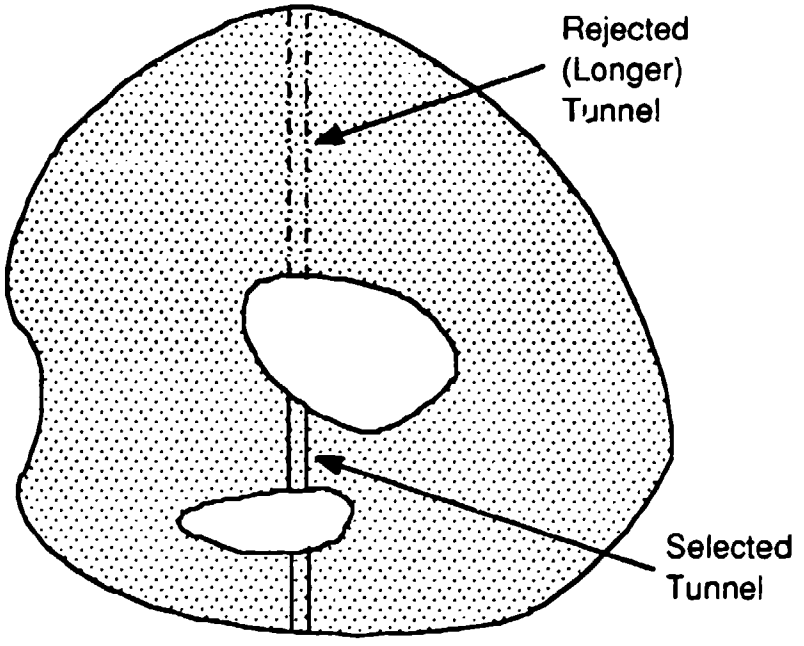


Figure 4-3. Tunnel Creation for Enclosed Holes.

Note that the tunnel passes through another hole on its way to the outside world. When this occurs, the second hole's region label is automatically equivalenced to 0; thus, no additional tunnel will be drawn for it.

The final action in the hole-processing procedure is returning the labels of all 0 pixels back to 0 so that the weather-region labeling procedure can be performed. By construction, this means that all labels greater than 1 must be reset to 0.

5. EXACT-REPRESENTATION ALGORITHM

For small weather regions, it is more bit efficient to simply specify the state of each pixel, 0 or 1, than it is to encode the region by an ellipse or polygon. Two shape specifications have been defined for such regions: a square and a rectangle. This section presents the algorithms that implement these shape representations.

5.1 EXACT SHAPE DEFINITION

In order to locate either a square or a rectangle on the map, the coordinates of the upper left corner of the figure and the dimensions of the figure must be provided. The advantage of the square over the rectangle is that only one length, not two, need be encoded. Thus it will occasionally be more efficient to enlarge the shorter dimension of a rectangular weather region to permit the use of the square shape representation. For example, a weather region whose pixels are contained within a 2x3 rectangle can be converted to a 3x3 square by "unnecessarily" including an additional 3 pixels within the region.

Rather than always having to compute whether a square or a rectangle is more desirable for a particular weather region, the algorithm adopted makes the following firm decision:

if the longer side of the weather region is ≤ 4 pixels,

use a square,

otherwise use a rectangle.

This rule has the added advantage of allowing us to save bits for the square representation, as now only 2 bits are required for its length field.

In actuality, only non-zero pixels contained within the weather region being represented need be encoded. If the encompassing square or rectangle covers part of another region, such as in the case of Figure 5-1, that second region will be encoded by its own shape.

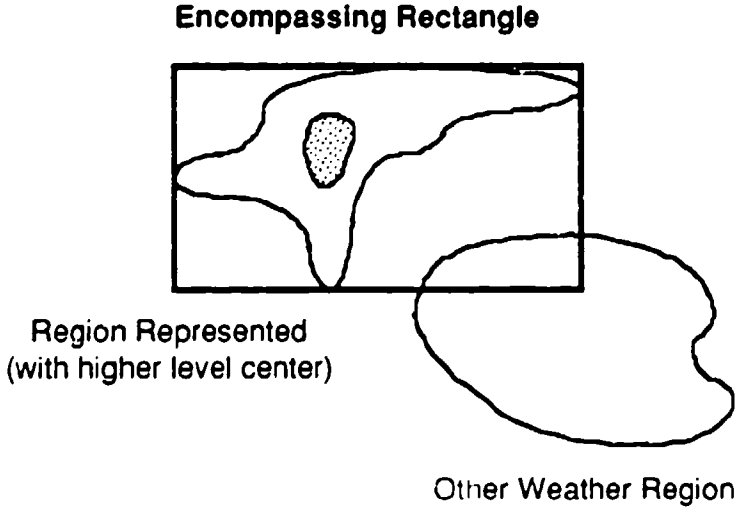


Figure 5-1. Encompassing Rectangle for a Weather Region.

However, since that other shape will in general not be exact, encoding all non-zero pixels found within the square or rectangle boundary will add to the fidelity of the decoded map.

Similarly, non-zero pixels that represent a higher weather level could be encountered within the exact-encoded weather region, as also indicated in Figure 5-1. In theory, encoding these pixels now will serve no useful purpose, as they should be encoded again and overwritten when their own level is processed. However, if that level's shape encoding were to miss any of these pixels, they could be left blank on the output map. Including them in the exact specification provides an answer closer to the true level than the level 0 that might be seen otherwise.

5.2 EXACT SHAPE ENCODING

The encoding logic for small weather regions is quite simple and straightforward:

1. Determine the minimum and maximum rows, r_{min} and r_{max}, of the weather region, and set $\Delta R = r_{max} - r_{min}$.
2. Determine the minimum and maximum columns, c_{min} and c_{max}, of the weather region, and set $\Delta C = c_{max} - c_{min}$.
3. Let $\Delta M = \text{Max}\{\Delta R, \Delta C\}$.
4. If $\Delta M \leq 4$, encode a square having a side of length ΔM,

   otherwise encode a rectangle having length ΔR and width ΔC.
5. Encode the coordinates r_{min} and c_{min}.
6. One by one, encode each of the pixels within the extent of the selected shape.

The rectangle length and width dimensions are encoded in 15 unit chunks, because then only 4 bits instead of 6 are required for the vast majority of situations. This implies that the number of bits required to encode these dimensions varies with the actual measurement extents. The details of the specific encoding logic for these shapes is provided by inference in Section 5.3.

5.3 EXACT SHAPE DECODING

Step 15 of the map-decoding procedure in Section 3.5 is used to determine if a square or rectangle has been encoded. The square shape decoding scheme, to read the dimension, location, and pixel values of the small (at most 4x4) weather region being represented, consists of the following steps:

1. Read V, the value of the first 2 message bits.
2. Set $\Delta M = V+1$.
3. Read V, the value of the next 6 message bits.
4. Set the upper row index $r_{min} = V+1$.
5. Read V, the value of the next 6 message bits.
6. Set the left column index $c_{min} = V+1$.
7. If $\Delta M = 1$, set the pixel at $(r_{min}, c_{min}) = 1$, stop.

8. Read the next $\Delta M * \Delta M$ message bits $b_1, b_2, \cdots, b_{\Delta M * \Delta M}$ and use them to set the values of the pixels in the square from (r_{min}, c_{min}) to $(r_{min+\Delta M}, c_{min+\Delta M})$.

Note in step 7 that the pixel in a 1x1 weather region must be a 1, so that it need not be encoded.

The decoding scheme for a rectangle is somewhat more complex since, as discussed above, its dimensions are given incrementally. The steps in this case are:

1. Set $\Delta R = 1$.
2. Read V, the value of the next 4 message bits.
3. Increase ΔR by V.
4. If $V < 15$, go to step 5; else if $V = 15$, return to step 2.
5. Set $\Delta C = 1$.
6. Read V, the value of the next 4 message bits.
7. Increase ΔC by V.
8. If $V < 15$, go to step 9; else if $V = 15$, return to step 6.
9. Read V, the value of the next 6 message bits.
10. Set the upper row index $r_{min} = V+1$.
11. Read V, the value of the next 6 message bits.
12. Set the left column index $c_{min} = V+1$.
13. Read the next $\Delta R * \Delta C$ message bits $b_1, b_2, \cdots, b_{\Delta R * \Delta C}$ and use them to set the values of the pixels in the rectangle from (r_{min}, c_{min}) to $(r_{min+\Delta R}, c_{min+\Delta C})$.

6. ELLIPSE-REPRESENTATION ALGORITHM

This section presents the various procedures that together constitute the algorithm for best representing a weather region by an ellipse. The ellipse that best represents a weather region is defined to be the one that matches the region's major shape parameters. In particular, the ellipse and the region will have the same:

1. Center x_0 , y_0
2. Area A
3. Orientation angle Θ of main axis
4. Ratio of along-axis to cross-axis moments of inertia L_a/L_c

The fourth point is equivalent to providing the same elongation or eccentricity for the ellipse and weather region.

6.1 WEATHER-REGION SHAPE PARAMETERS

The parameters of the weather region are determined by scanning the single-level map row by row to determine the pixels included within the region. Let:

$$p_{ij} = \begin{cases} 1 & \text{if row i col j is in the region} \\ 0 & \text{otherwise} \end{cases}$$

Then the row, column, and diagonal sums of the region are given by:

$$R_i = \sum_{j=1}^{n} p_{i\,j} \qquad i = 1,n$$

$$C_j = \sum_{i=1}^{n} p_{i\,j} \qquad j = 1,n$$

$$D_k = \sum_{i=1}^{n} p_{i\,;k-i} \qquad k = 2,2*n$$

The area and center of the region can then be computed directly from these sums as follows:

$$A = \sum_{i=1}^{n} R_i$$

$$x_0 = \frac{1}{A} \sum_{i=1}^{n} i*R_i$$

$$y_0 = \frac{1}{A} \sum_{j=1}^{n} j*C_j$$

Next, the orientation angle of the region's major axis is calculated by the following set of steps:

$$s1 = \sum_{i=1}^{n} i^2*R_i$$

$$s2 = \sum_{j=1}^{n} j^2 \cdot C_j$$

$$s3 = \sum_{k=2}^{2 \cdot n} k^2 \cdot D_k$$

$$a = s1 - A \cdot x_0^2$$

$$c = s2 - A \cdot y_0^2$$

$$b = \frac{1}{2}\left[s3 - (s1 + s2) \right] - A \cdot x_0 \cdot y_0$$

$$\Theta = \frac{1}{2} \operatorname{atan}\left(\frac{2 \cdot b}{a-c}\right)$$

Finally, the along-axis and cross-axis moments of inertia of the region are given by:

$$L_a = \frac{a+c}{2} - \frac{\sqrt{(2b)^2 + (a-c)^2}}{2}$$

$$L_c = \frac{a+c}{2} + \frac{\sqrt{(2b)^2 + (a-c)^2}}{2}$$

6.2 ELLIPSE PARAMETERS

The definitions of the ellipse parameters for a typical ellipse located on the map coordinate system are illustrated in Figure 6-1.

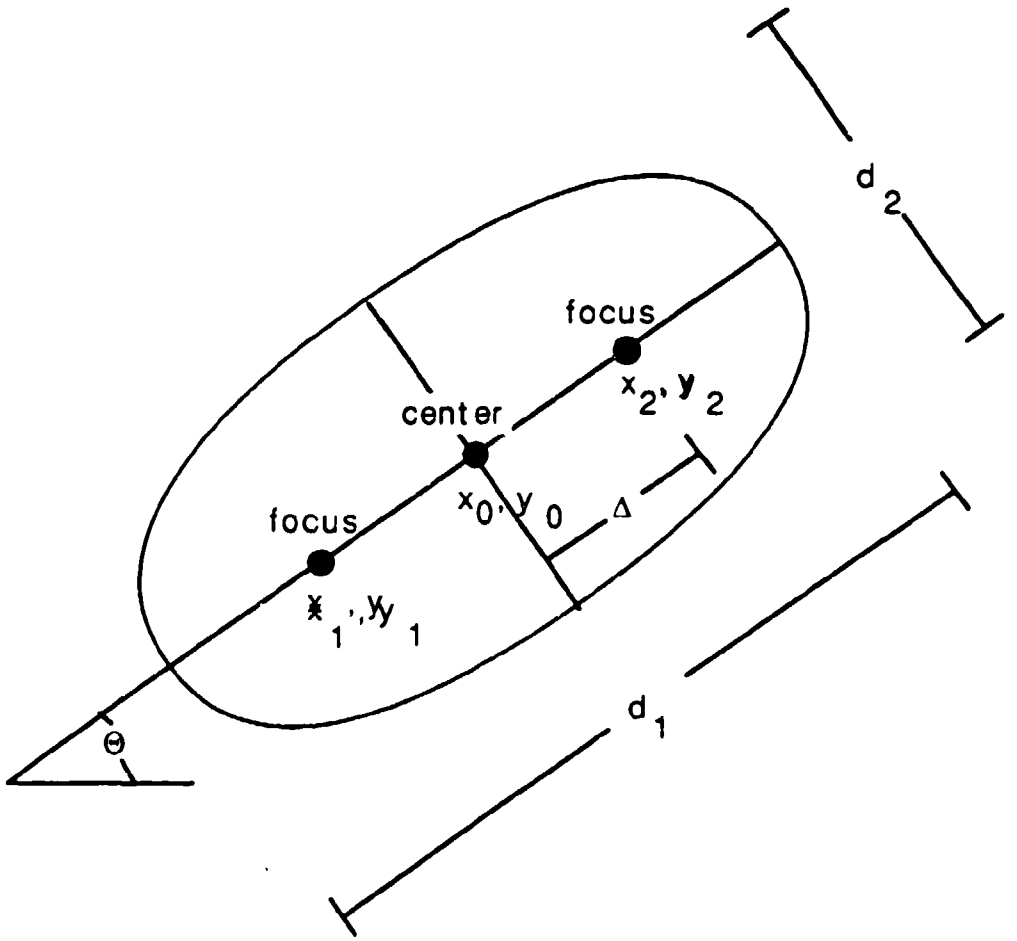


Figure 6-1. Ellipse Parameter Specifications.

As seen, the parameters include the locations of the ellipse's center and its two foci, the lengths of its major and minor axes, and its orientation angle.

The ellipse can be mathematically defined as the locus of all points x,y satisfying:

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} + \sqrt{(x-x_2)^2 + (y-y_2)^2} = d_1$$

In words, an ellipse is the set of points the sum of whose distances from two fixed points is a constant; the two points being the foci of the eilipse, and the distance sum being the length of its major axis.

6.3 "ELLIPSETICITY" FACTOR

The formula for the area of an ellipse is the generalization of the formula for the area of a circle:

$$A = \pi \frac{d_1}{2} \frac{d_2}{2} \tag{1}$$

Similarly, the along-axis and cross-axis moments of inertia are the circle generalizations:

$$L_a = A \frac{1}{4} \left(\frac{d_2}{2}\right)^2 \tag{2}$$

$$L_c = A \frac{1}{4} \left(\frac{d_1}{2}\right)^2 \tag{3}$$

Note that the product of the moments of inertia can be expressed in terms of the area as follows:

$$L_a * L_c = \frac{A^4}{16\pi^2} \tag{4}$$

Now define a factor to be named the area-to-inertia factor or more simply the k-factor:

$$k = \frac{A}{2\sqrt{\pi \sqrt{L_a * L_c}}} \tag{5}$$

This factor is clearly 1.0 for an ellipse by the way it was defined using (2) and (3). Since the ellipse has the largest area relative to its moments of inertia for any geometric figure, any other shaped region will have a k-factor less than 1.0. In particular, from (5), we can write:

$$A = 2\sqrt{\pi \sqrt{k^2 L_a * k^2 L_c}}$$

and thus conclude that for an ellipse and a general region to have the same area and elongation factor (ratio of moments of inertia), the moments of inertia of the ellipse must be a factor of k^2 smaller than those of the region.

In summary, the magnitude of a region's k-factor can be used to measure its ellipseticity. This fact is used in the shape-fitting algorithms in deciding how realistically an ellipse can represent a given weather region. In particular, Figure 6-2 illustrates the "ellipseticity" factor for the four regions on a simple weather map. It is clear from this figure how a reduction in k-factor matches the reduction in perceived ellipseticity. In the PE algorithm, the region whose k-factor is 0.98 will always be represented by an ellipse; the other regions will be represented by a polygon if sufficient bits exist.

6.4 ELLIPSE FITTING

With this background, it is now possible to calculate the parameters x_1, y_1, x_2, y_2, and d_1 of the ellipse that, by the stated criteria, best represents a weather region. First, the ellipse and region must have the same center:

$$x_0 = x_{region}$$

$$y_0 = y_{region}$$

Red Regions (Top - Bottom): 0.93 0.98 0.96 Green Region: 0.88

Figure 6 2. "ELlipselicity" Factors of Weather Regions.

Then, the lengths of the ellipse axes can be computed by (3) and (4) and the interpretation of the k factor to be:

$$d_1 = 4 * k * \sqrt{L_c / A}$$

$$d_2 = 4 * k * \sqrt{L_a / A}$$

Since the endpoint of the minor axis is on the ellipse as depicted in Figure 6-1, it must satisfy the ellipse equation. Thus, by symmetry:

$$\left(\frac{d_2}{2}\right)^2 + \Delta^2 = \left(\frac{d_1}{2}\right)^2, \text{ or}$$

$$\Delta = \sqrt{\left(\frac{d_1}{2}\right)^2 - \left(\frac{d_2}{2}\right)^2} \qquad (6)$$

Finally, the foci are located at a distance Δ from the center along the region's major axis, which is at the previously computed orientation angle Θ:

$$x_1 = x_0 - \Delta * \cos(\Theta)$$

$$y_1 = y_0 - \Delta * \sin(\Theta)$$

$$x_2 = x_0 + \Delta * \cos(\Theta)$$

$$y_2 = y_0 + \Delta * \sin(\Theta)$$

6.5 ELLIPSE REPRESENTATION

The simplest and most compact method for defining an ellipse is to specify the locations of its two foci and the length of its characteristic distance d. The problem that arises when this approach is used for ellipse encoding is that these parameters are decimal numbers, and only integers (or scaled integers) can be transmitted in a bit string.

The most direct resolution of this problem is to round off the defining values to the nearest integer. Unfortunately, the shape of an ellipse can be very strongly affected by small adjustments to its parameters; in the worst case, the ellipse can even disappear when roundoff is applied. The solution adopted to alleviate this effect is to round off the foci coordinates but then recompute a new distance d that, combined with these new foci locations, produces an ellipse having the same area as the exactly calculated ellipse. This modified value of d is then rounded to the nearest quarter integer.

After rounding the foci coordinates, the new distance Δ of the modified ellipse becomes:

$$\Delta = \frac{1}{2} * \sqrt{(x_{2r} - x_{1r})^2 + (y_{2r} - y_{1r})^2}$$

where the r subscript indicates rounded values. Now applying formulas (1) and (6), we see that the area can be expressed as:

$$A = \pi * \frac{d}{2} * \sqrt{\left(\frac{d}{2}\right)^2 - \Delta^2}$$

Finally, solving for the modified distance d:

$$d_{mod} = \sqrt{2 \cdot \Delta^2 + \frac{2}{\pi}\sqrt{\pi^2 \cdot \Delta^4 + 4 \cdot A}} \qquad (7)$$

Figure 6-3 illustrates the need for recalculating the distance parameter when roundoff is employed. The ellipse in the upper left is produced by utilizing the exact ellipse parameters, namely

focus 1 at location 12.3 by 12.3

focus 2 at location 48.7 by 48.7

distance d = 51.7

When these values are rounded off to 12, 49, and 52, respectively, the ellipse produced, as shown at the upper right, is null. The recalculation of equation (7) modifies the distance d to the adjusted value of d_{mod} = 52.54. The lower left and right parts of the figure present the ellipses produced by rounding this new value of d to an integer (54) and a scaled integer (54.50), respectively. Clearly the latter ellipse is the only one that closely matches the originally specified picture.

6.6 ELLIPSE ENCODING

As discussed in Section 6.5, encoding an ellipse consists of transmitting 5 parameter values. For a 64x64-pixel map, having a maximum diagonal of 91, the number of bits required for a straightforward representation of these values would be as follows:

focus 1, x coordinate	1-64 (integer)	6
focus 1, y coordinate	1-64 (integer)	6
focus 2, x coordinate	1-64 (integer)	6
focus 2, y coordinate	1-64 (integer)	6
distance	0-91 (1/4 integer)	9
		33

Several bit-reduction "tricks" are employed in the data compression program to significantly reduce this requirement. First, the ellipses on any given weather level are transmitted in decreasing order of their distance parameter d. Thus, the largest possible value of the current ellipse's d is the value of the previous ellipse's d. If, for example, the list of successive distances are as follows, the number of bits required results in a savings of 8 bits, or 22%.

d	bits needed to encode
48.75	9
14.50	8
5.00	6
5.00	5

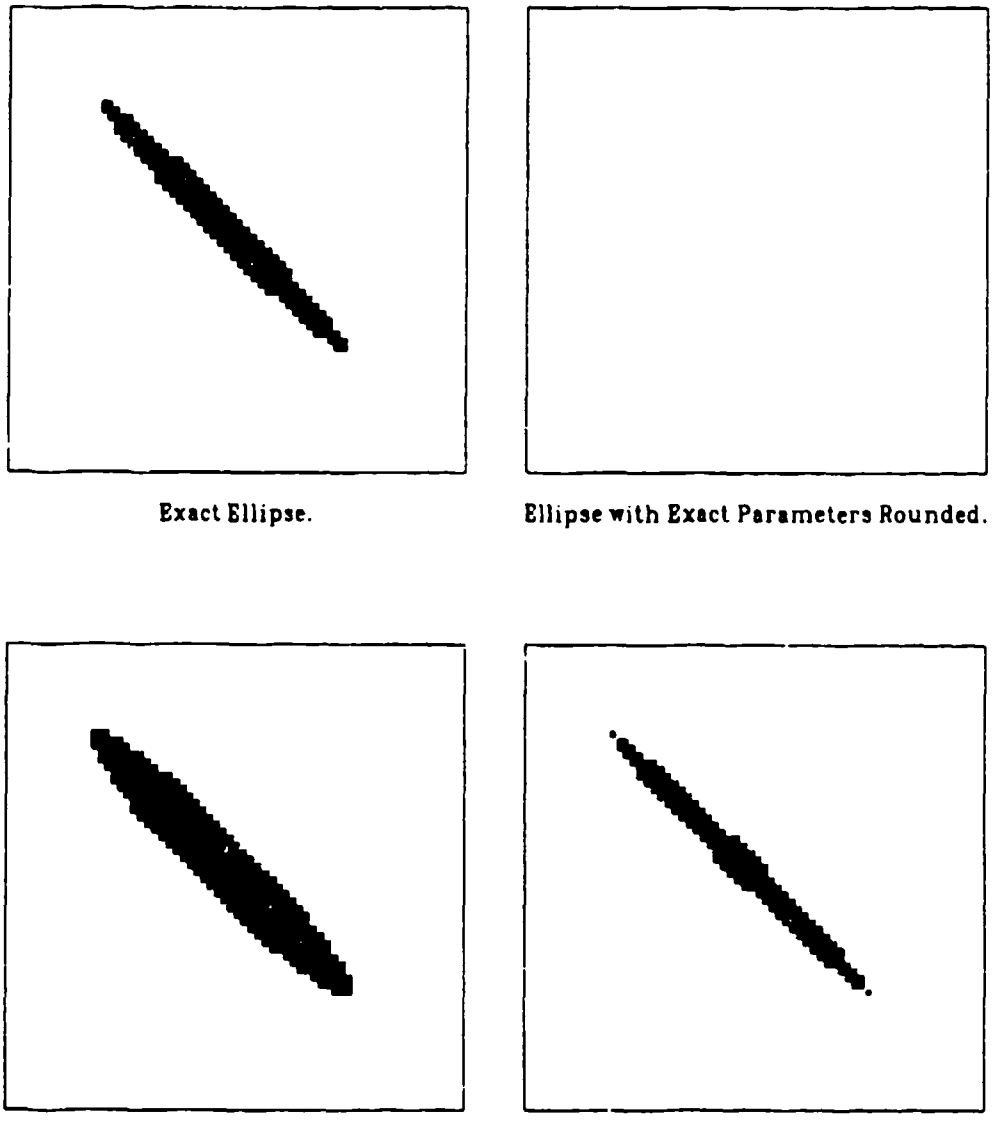


Figure 6-3. *Effect on Ellipse of Rounding its Parameters.*

Next, the foci are ordered so that $x_1 \le x_2$, and x_1 and y_1 of the first focus are encoded directly using the full 6 bits. Then the incremental value x_2-x_1 is considered for encoding. Since x_2 must be greater than x_1 by the ordering method, fewer than 6 bits will be required to represent the increment whenever the first focus is on the lower half of the map; in the best case of both foci on the lowest map row, no bits at all will be required to encode x_2.

The final bit-saving method, employed for encoding the remaining value y_2, makes use of the fact that the foci can be separated by at most the ellipse distance d (producing a straight line "ellipse"). Thus:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \le d$$

from which it follows that

$$y_{2min} = \text{Max} \left\{ 1, y_1 - \sqrt{d^2 - (x_2 - x_1)^2} \right\}$$

$$y_{2max} = \text{Min} \left\{ 64, y_1 + \sqrt{d^2 - (x_2 - x_1)^2} \right\}$$

Therefore, the possible set of values for y_2 will often be restricted to a very small range. By computing the extent of this range, the number of bits required to encode the value can be determined. This number of bits is then used to encode the difference (y_2 - y_{2min}).

6.7 ELLIPSE DECODING

The ellipse-decoding algorithm, to read from the encoded bit stream the ellipse parameters x_1 and y_1 of focus 1, x_2 and y_2 of focus 2, and the distance d, is the inverse of the above encoding scheme. Since the decoder will know the tricks used by the encoder, it can generate the proper values by the following steps:

1. Set $d_{previous}$ to 100.
2. Compute b = $\log_2(4 * d_{previous})$ rounded up to next integer.
3. Read V, the value of the next b message bits.
4. Set d = V/4.
5. Set $d_{previous}$ = d.
6. Read V, the value of the next 6 message bits; set x_1 = V+1.
7. Read V, the value of the next 6 message bits; set y_1 = V+1.
8. Compute b = $\log_2(64-x_1+1)$ rounded up to next integer.
9. If b>0,

   read V, the value of the next b bits; set $x_2 = x_1 + V$.

   Else set $x_2 = x_1$.
10. Using d, x_1, x_2, and y_1, compute y_{2min} and y_{2max}.
11. Compute b = $\log_2(y_{2max} - y_{2min} + 1)$ rounded up to next integer.

12. If b>0,

    read V, the value of the next b bits; set $y_2 = y_{2min} + V$.

    Else set $y_2 = y_1$.

13. Return to step 2 if more ellipses exist for this level.

6.8 ELLIPSE-FILL PROCEDURE

Once the 5 ellipse parameters x_1 and y_1 of focus 1, x_2 and y_2 of focus 2, and the distance d have been decoded, the map fill procedure can be performed in a very straightforward manner: scan the map pixel by pixel and fill in each pixel x, y for which

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} + \sqrt{(x-x_2)^2 + (y-y_2)^2} \leq d \qquad (8)$$

However, this scanning procedure can be accelerated considerably with two modifications.

The first modification is to limit the search region to a square guaranteed to encompass all the ellipse fill points. Clearly, using (8), any filled point must satisfy:

$$|x-x_1| + |x-x_2| \leq d$$

Thus the x extent of the fill region must be limited to:

$$\frac{x_1 + x_2 - d}{2} \leq x \leq \frac{x_1 + x_2 + d}{2} \qquad (9)$$

Similarly, the y extent of the fill region must be limited to:

$$\frac{y_1 + y_2 - d}{2} \leq y \leq \frac{y_1 + y_2 + d}{2} \qquad (10)$$

The second modification makes use of the property that ellipses have no interior holes. Thus, on a given row, filled points will extend consecutively from the first to the last. This fact leads to the following minimum time ellipse fill algorithm:

1. Set x to x_{min} as determined from (9).
2. Set y to y_{min} as determined from (10).
3. Increment y until y_a, the first value for which (8) is satisfied.
4. Set y to y_{max} as determined from (10).
5. Decrement y until y_b, the first value for which (8) is satisfied.
6. Fill in row x from y_a to y_b inclusive.
7. Increment x by 1.
8. If $x > x_{max}$ as determined from (9), stop;

   Else return to step 2.

7. POLYGON-REPRESENTATION ALGORITHM

This section presents the various procedures that together constitute the algorithm for optimally representing a weather region by a polygon, subject to stated constraints. The definition of *optimal* is, as expected, the best match between the actual weather region and the interior of the polygon. The constraints, which all translate into distortion of the "perfect" polygon through reduction in the number of its vertices, are required to meet the overall data-link message-bit limit.

7.1 WEATHER-REGION TRACING

The process of producing the polygon that best encompasses a given weather region has two steps. First, the contour of the region is traced, creating an ordered list of edge points. Then successive points that are co-linear are joined together to form the line segments that become the sides of the polygon. The parameters selected to define the meaning of *co-linear* implement a tradeoff between fidelity of fit and number of polygon sides, and can only be set empirically.

The tracing step traverses the boundary of the weather region in a clockwise manner. It does this by employing the "leftmost" rule: whenever the current point has more than one neighbor in the region, select the next one to be the one that is the leftmost relative to the last direction of motion. Figure 7-1 shows the order of search when the last direction was non-diagonal and diagonal, respectively. The indicated illegal directions indicate that there are places where region pixels cannot reside; had these neighbors been in the region, they would have been traversed prior to the current point, and the direction of entry to the current point would not have been that shown.

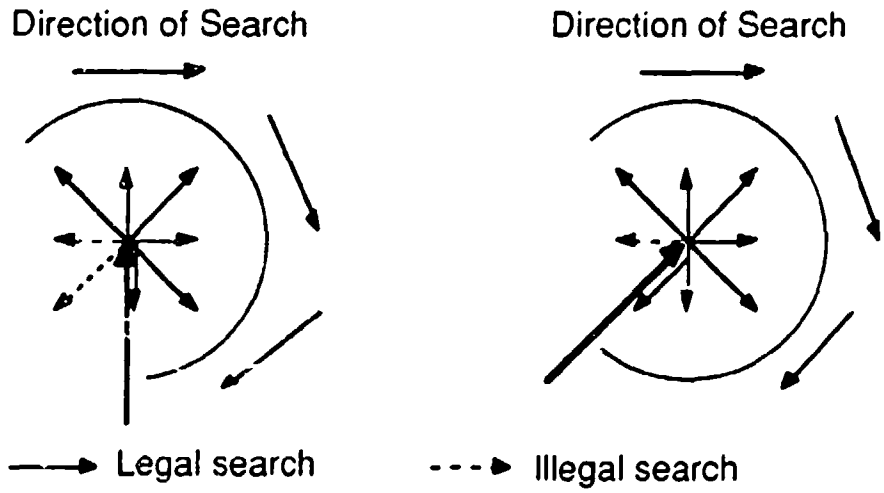


Figure 7-1. Search Pattern for Locating Next Contour Pixel.

Some examples of the use of this leftmost rule in tracing part of a contour boundary are seen in Figure 7-2. Note that a doubling back action is required when an isolated point is reached. Also, note that a point can be entered more than once, but that the exit direction will differ each time because the entrance direction will be different.

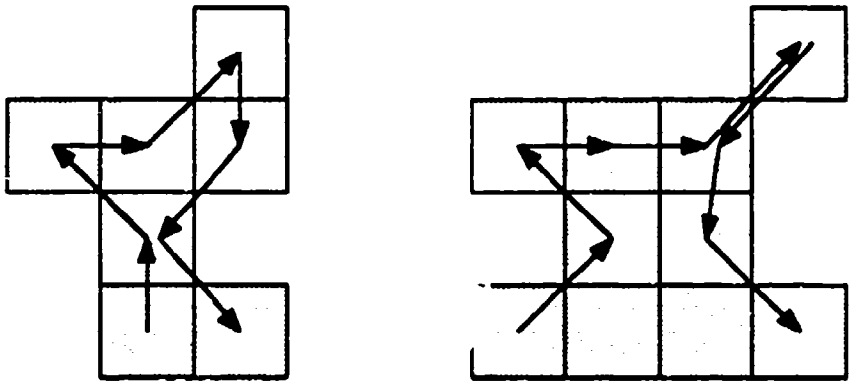


Figure 7-2. Part of Contour Tracing of Sample Regions.

The tracing of a weather region is initiated at the "upper left" point of the region, defined as the leftmost point on the uppermost row of the region (even though visually the point could be at the upper right of the region). The initial direction of search is set to "right." The tracing is completed when the sequence of first point followed by second point again appears; just returning to the first point is not a sufficient test because of the possibility of multiple entrances as noted above.

7.2 INITIAL POLYGON CONSTRUCTION

The second step of the polygon formation process is the grouping of co-linear boundary points into the line segments that become the polygon sides. A simple approach would be to add points one by one to the current grouping until the co-linear conditions were first violated. However, this one-at-a-time method would be very time consuming. Instead, a set of n+1 successive points (nominally six) are considered at one time. This set is tested for co-linearity, and the following steps taken based on the result:

1. If the set is co-linear, create a new line segment that extends from the first to the last point and proceed to step 3.
2. Otherwise, delete points from the set, starting at the end, to create a smaller set, and return to step 1.
3. Compare the slope of the new line segment to that of the previous line segment.
4. If the slopes are compatible, join the segments into a longer single segment, otherwise leave them separate.
5. Create a new candidate set of n+1 points, consisting of point 0 being the last point used to form the newest line segment, and the next n points, and return to step 1.

The test for co-linearity is defined as follows. First, form the line segment that connects the first (0) and last (n) points in the set. Then, for each point in the set, calculate the distance d_i from

that point to the nearest part of the line segment. Finally, the set is co-linear if the following two conditions are met:

1. $d_i \leq D_1$ for all i in the set, and
2. $\sum_{i=0}^{n} d_i \leq D_2$

D_1 and D_2 are parameters, nominally set to 1.0 and 2.0, respectively. Figure 7-3 indicates how the values of d_i are defined for various geometric cases, where the heavy line is the reference line segment.

Figure 7-3. Examples of Calculation of Distance d_i.

Note that a point can lie on the extension of the line segment, but because it is "off the end" of the segment, its d_i is non-zero. Thus, the procedure for computing d_i is more complex than just using the standard formula for the distance from a point to a line:

$$d_{line} = \frac{|A \cdot x_i + B \cdot y_i + C|}{L}$$

where A, B, and C specify the equation of the line:

$$A \cdot x + B \cdot y + C = 0$$

$$A = y_0 - y_n$$

$$B = x_n - x_0$$

$$C = x_0 \cdot y_n - x_n \cdot y_0$$

and $$L = \sqrt{A^2 + B^2}$$

Instead, the distances d_0 and d_n of the point from each end of the line segment must also be computed:

$$d_0 = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}$$

$$d_n = \sqrt{(x_i - x_n)^2 + (y_i - y_n)^2}$$

Then the distance is determined according to the applicable case:

$$d_i = \begin{cases} d_0 & \text{if } d_n^2 > (d_0^2 + L^2) \\ d_n & \text{if } d_0^2 > (d_n^2 + L^2) \\ d_{line} & \text{otherwise} \end{cases}$$

This co-linearity test would appear to be very time consuming. Fortunately, all the work can be performed at program initialization and the results stored in a lookup table. This action is possible because the co-linearity result is totally defined by the set of n directions of movement that connect the n+1 points. Thus, at program startup, all 8^n permutations of movement are tested for co-linearity by the above procedure; if co-linearity is obtained, the value 0 is stored in the corresponding table entry, otherwise, the index of the point i whose distance d_i is greatest is stored. At run time, the n actual directions of movement are used to construct an index into the lookup table. If the table entry is 0, co-linearity is known to exist for the n+1 points; otherwise, if the entry is m, the m+1 initial points of the set are used in step 2 of the above algorithm as the new set to be tested by another table lookup.

Once a new co-linear line segment is identified, the decision must be made as to whether or not to join it with the previous segment to form a longer segment. The new and old segments are compatible and can be joined if their slopes are similar enough. The angular difference θ between any two line segments is given by:

$$\cos\theta = \frac{A_1 * A_2 + B_1 * B_2}{L_1 * L_2}$$

If θ_0 and θ_{now} are, respectively, the slope of the previous segment when it was first formed and the slope of the previous segment at the current time (after segment joinings, if any, have occurred), and θ_i is the slope of the new segment, then the segments are compatible and can be joined if:

$$\cos|\theta_i - \theta_{now}| \leq \cos T_1 \quad \text{and}$$

$$\cos|\theta_i - \theta_0| \leq \cos T_2$$

where T_1 and T_2 are parameters nominally set to 20 and 30 degrees, respectively. In other words, the tests say that the new segment cannot differ from the previous one by more than T_1 degrees. Furthermore, if previous joinings have already "bent" the previous line segment, the new segment cannot differ by more than T_2 degrees from its original direction.

Finally, after all boundary points have been processed into line segments, the set of segments that then exist constitute the polygon that best represents the weather region. The set of end points of these segments are then taken as the list of polygon vertices that would be transmitted to the user if bit limitations were not present. If, however, bit reduction is required, the vertex reduction algorithm of Section 7.3 is employed.

7.3 POLYGON VERTEX-REDUCTION ALGORITHM

Two related procedures are employed to reduce the number of vertices required to specify a region-contouring polygon when bit limitations are present. The simplest one is just to remove a vertex, while the more complex one is to replace two successive vertices by the single vertex that best

matches their contouring effect. In either case, the vertex to select is the one whose removal (or replacement) introduces the least contour distortion.

Distortion, for either approach, is measured by the amount of error, either overfill (pixels added to the true weather region) and/or underfill (pixels removed from the region), added to the polygon approximation as a result of the vertex reduction. Since an actual calculation of distortion would require performing a polygon fill procedure for each existing vertex, an approximate method has been adopted: assume the present polygon is a perfect match to the region and calculate the change made by the removal of each vertex. This procedure only necessitates finding the area of triangles, and as such is much quicker.

The distortion area that results from the removal of a single vertex i is seen in Figure 7-4.

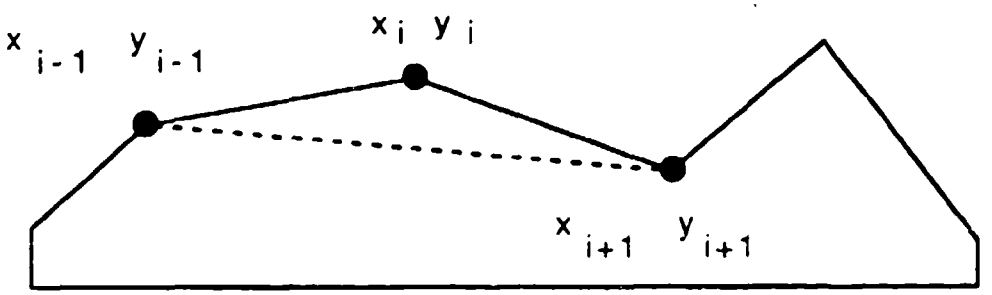


Figure 7-4. Effect of Removing Vertex i.

The formula of the area removed when the dotted line is substituted for the two segments is:

$$A = 1/2 * | (x_i - x_{i-1}) * (y_{i+1} - y_{i-1}) - (x_{i+1} - x_{i-1}) * (y_i - y_{i-1}) |$$

The loss of this area can result in the creation of either an underfill or an overfill. The determining factor is whether the angle at vertex i is a convex (inward, as at vertices i-1 and i in the above figure) or concave (outward, as at vertex i+1 in the figure) angle of the polygon, respectively. If the polygon is traced clockwise, the angle at vertex i is concave if:

$$(x_i - x_{i-1}) * (y_{i+1} - y_i) - (x_{i+1} - x_i) * (y_i - y_{i-1}) > 0$$

and convex otherwise. Finally, creating an underfill is a more serious effect than is creating an overfill, as underfill results in real weather going unreported. Thus the error "charged" to a single vertex for its removal has been empirically set as follows:

$$E_1 = \begin{cases} A & \text{if vertex is convex} \\ A/2 & \text{if vertex is concave} \end{cases}$$

The calculation of distortion for the replacement approach is more complex. The method of selecting the new vertex with which to replace a successive pair of existing vertices depends upon the convex or concave nature of the two vertex angles. If both angles are convex, the optimum point, in terms of minimum distortion introduced, will be located as seen in Figure 7-5.

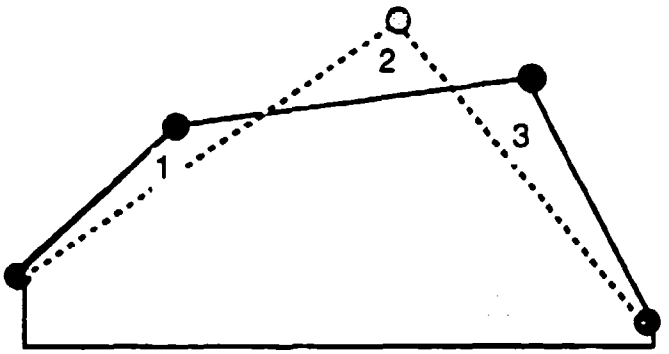


Figure 7-5. Replacement Vertex for Two Convex Vertices.

Since the calculation of the exact coordinates of the optimum replacement point is quite complex, a reasonable approximation is made instead. Draw lines from the two end vertices through the quarter points of the middle line; the intersection of these lines is the selected point. The error area produced by the two-to-one vertex replacement then consists of the three triangles evident from Figure 7-5. Computing the areas of each triangle by the above formula, and noting that triangle 2 in the figure is an overfill while the others are underfills, the total error area becomes:

$$E_2 = A_1 + A_2/2 + A_3$$

When one of the vertex angles is concave, the optimum replacement point calculation will differ considerably from the above case. Figure 7-6 indicates a reasonable approximation as the center of the middle line.

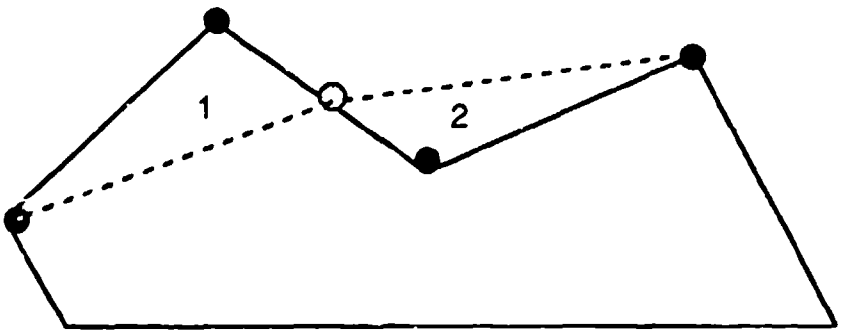


Figure 7-6. Replacement Vertex for One Convex and One Concave Vertex.

This replacement geometry produces an error area consisting of two triangles, an underfill (1) corresponding to the convex vertex and an overfill (2) corresponding to the concave one. Thus, the total error area in this case is:

$$E_2 = A_1 + A_2/2$$

Finally, if both angles are concave, the geometry of the first case could again be employed, this time producing two overfill and one underfill triangle. However, the more critical underfill error area can be eliminated by using the centerpoint of the middle line as done in the second case, producing the geometry of Figure 7-7.

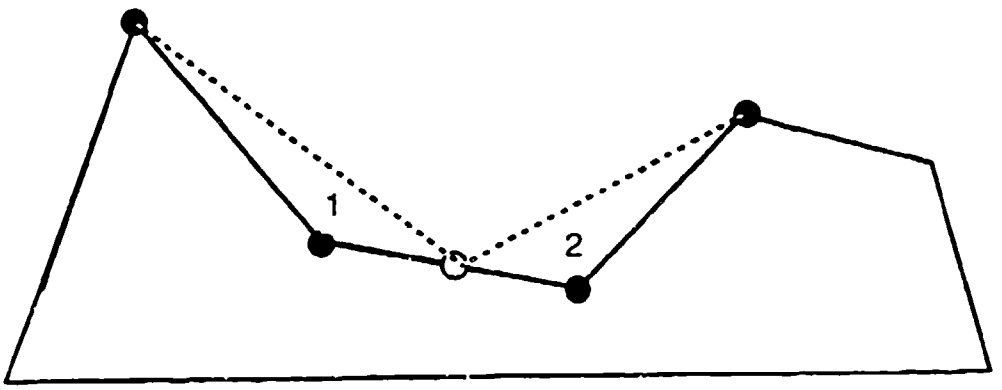


Figure 7-7. Replacement Vertex for Two Concave Vertices.

The total error area in this case then becomes:

$$E_2 = (A_1 + A_2)/2$$

The vertex reduction algorithm proceeds, at each step, by performing the action, either vertex removal or vertex pair replacement, that produces the minimum error area. The overall algorithm consists of the following steps:

1. Calculate the removal and replacement areas, E_{1i} and E_{2i}, for each vertex i of the polygon.
2. Select the smallest value E_{min} from the combined set of E_1 and E_2 entries.
3. Perform the indicated removal ($E_{min} \in E_1$) or replacement ($E_{min} \in E_2$) to produce the vertex-reduced polygon.
4. If the reduction criterion has now been satisfied. stop.
5. Otherwise, recompute the values of E_1 and E_2 that have been changed by the vertex reduction and return to step 2.

For step 5, only the neighboring vertices of the one removed (or two replaced) need error recomputations; the geometry of the other points remain unchanged.

The reduction criterion referred to in step 4 is the maximum percent distortion that the polygon will be permitted to undergo. Thus if E_j is the error selected at reduction stage j, the process is terminated after s-1 stages when E_s would violate one of the following conditions:

$$\sum_{j=1}^{s} E_j \leq D_1 * \text{weather region area}$$

$$E_s \leq D_2 * \text{weather region area}$$

That is, the total error must not exceed D_1 percent, and no error may on its own exceed D_2 percent of the original weather region area. D_1 is the distortion parameter set according to the current pass number of the level being processed (see Section 3), while D_2 is a constant empirically set at 10%.

Figure 7-8 illustrates the successive polygon approximations for a typical weather region. The first subfigure is the traced shape of the weather region. The second subfigure is the best polygon fit for the region, with no vertex reduction. This polygon contains 58 vertices, and appears to match the region to a very high degree of accuracy. Finally, the last two subfigures illustrate the polygon after 10- and 20-percent error reductions; the number of vertices have been reduced to 16 and 8, respectively. Subjectively, the 10-percent reduction is still a reasonable approximation to the weather region, while the 20-percent reduction shows significant loss of fidelity.

7.4 POLYGON PRESENTATION

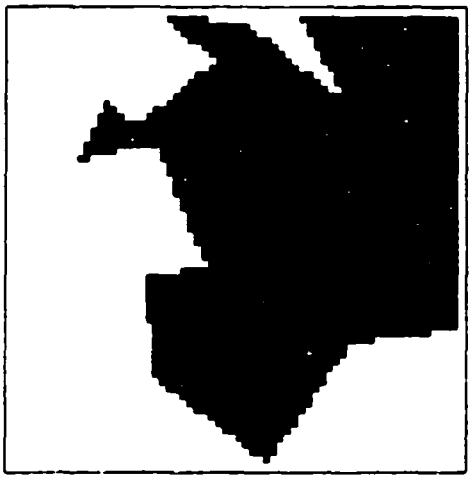
The obvious method for presenting a polygon is to list in order the coordinates of its vertices. The decoder then merely has to plot the specified points and "connect the dots" to produce the polygon contour. It would then determine the interior pixels of the polygon by some type of line-crossing procedure and fill in each point so found.

Many polygons, particularly the "normal" convex polygons, have properties that permit a much simpler fill procedure to be employed. As demonstrated in Section 7.7, the fill procedure for such polygons is essentially complete once the polygon contour has been drawn.

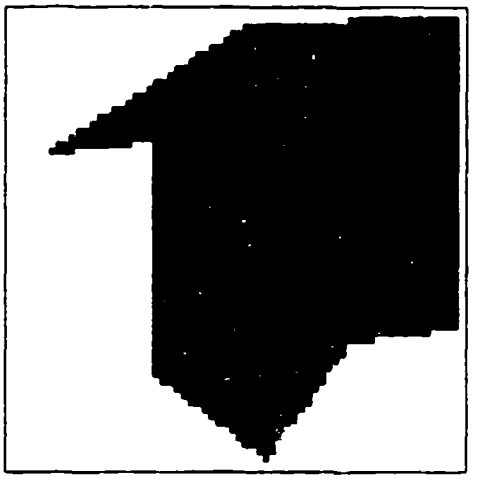
A polygon in reality does not have to be convex to satisfy the requirements of the simple routine. Instead, two types of simple polygons have been defined, "x-fillable" and "y-fillable." The definition of an x-fillable polygon is one that has at most two intersections with any line x=constant; the y-fillable definition is the converse. Figure 7-9 shows examples of non-convex x-fillable and y-fillable polygons, where the lines producing the violations in the other directions are as indicated.

Polygon Tracing.

Best Polygon Fit – 0% Distortion.



Polygon Fit with 10% Distortion.



Polygon Fit with 20% Distortion.

Figure 7–8. Polygon Approximations with Increased Distortion.

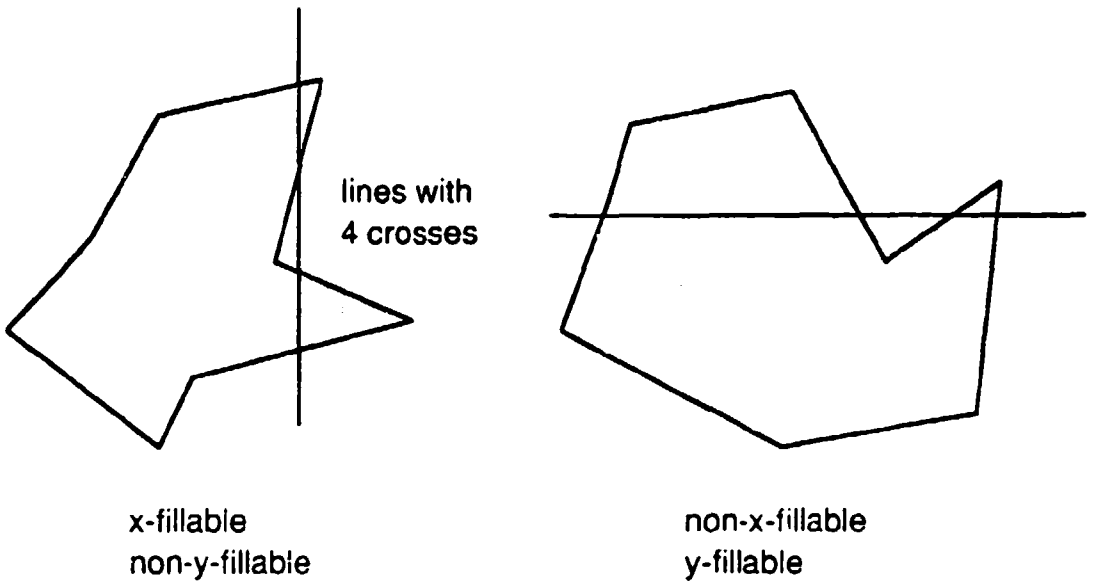


Figure 7-9. Non-Convex Fillable Polygons.

The algorithm for determining whether a polygon is x-fillable from its vertex list is as follows, where n vertices are assumed and x_i, y_i are the coordinates of vertex i:

1. Let (x_m , y_m) be the minimum x vertex, defined by

   $$x_m = \underset{i}{\text{Min}} \{ x_i \}$$

2. Let (x_r , y_r) be the maximum x vertex, defined by

   $$x_r = \underset{i}{\text{Max}} \{ x_i \}$$

3. Then the polygon is x-fillable if

   $x_i \leq x_{i+1}$ $\quad i = m, m+1, \ldots, r-1$ $\quad$ all mod n

   $x_i \geq x_{i+1}$ $\quad i = r, r+1, \ldots, m-1$ $\quad$ all mod n

That is, the polygon is x-fillable if the vertices are always non-decreasing "down the right side" and always non-increasing "up the left side." The definition of y-fillable is the same, with y substituted for x.

Normal polygons also have another property that can simplify the polygon presentation process. Namely, for many of the fillable class of polygons, the vertices can be listed in various shuffled orders, and the decoder can still generate the proper contour if it knows the shuffling algorithm. This fact is clearly true for a convex polygon, in which any random ordering of the vertices is acceptable to the decoder, but convexity is not a necessary condition. The following set of conditions, termed x-reorderable, will permit the n vertices of a polygon to be shuffled in a manner that facilitates bit reduction by the encoder:

1. The polygon must be x-fillable (see above).

2. Let s_0 be the reference slope, defined by

$$s_0 = \frac{y_r - y_m}{x_r - x_m}$$

3. The vertices down the right side of the polygon, from vertex m+1 to vertex r, must satisfy:

$$\frac{y_i - y_m}{x_i - x_m} > s_0$$

4. The vertices up the left side of the polygon, from vertex r+1 to vertex m, must satisfy

$$\frac{y_i - y_m}{x_i - x_m} < s_0$$

The x-fillable condition, as before, states that the polygon vertices cannot double back in the x direction, while the slope conditions state that all vertices on each side of the polygon must be located on their "own side" of the line connecting the minimum and maximum x-coordinate vertices. A sample x-reorderable polygon illustrates this in Figure 7-10.

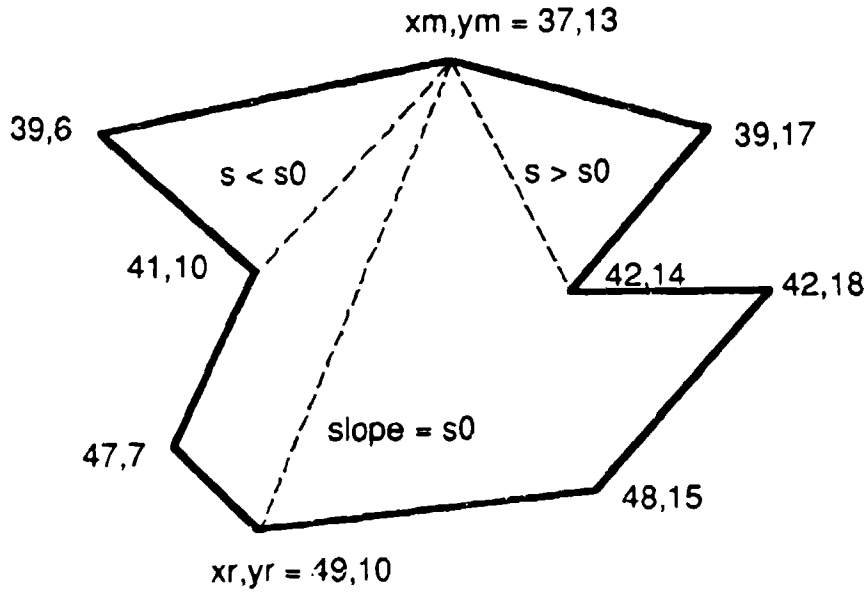


Figure 7-10. Example X-Reorderable Polygon.

A similar set of conditions is used to define a y-reorderable polygon. A polygon reorderable in one coordinate can be non-reorderable in the other; in fact, the figure above is non-y-reorderable because of doubling back and because the vertex at (42,14) violates the y slope condition. Section 7.5 illustrates how reorderable polygons are exploited to reduce encoding bit requirements.

7.5 POLYGON ENCODING

As discussed in Section 7.4, encoding a polygon consists of transmitting the number of polygon vertices n, and then the x, y coordinate pairs of these n vertices. For a 64x64 map, the number of bits required for a straightforward presentation of the coordinate values would be 6•2•n. However, when the polygon is reorderable by the above definition, this total can often be significantly reduced by a coding "trick."

With an x-reorderable polygon, the "left" and "right" vertex lists can be merged into a single list ordered by increasing x coordinate. The decoder will be able to properly unsort this list into its two components by making a slope test on each vertex. Thus, the order of vertex encoding employed for a x-reorderable polygon is:

x_m , y_m

x_r , y_r

x_i , y_i $\quad i \neq m,r$ ordered such that $x_i \geq x_{i-1}$

With this ordering, each coordinate x_i is guaranteed to fall into the interval $[x_{i-1}, x_r]$. This often permits fewer than 6 bits to be required for its encoding (no bits needed at all if $x_{i-1} = x_r$). For example, the x-coordinate bit requirements for the above example polygon is reduced as seen in Table 6 for a savings of 19 bits, or 35%.

Table 6. Encoding Requirements for Figure 7-10 Polygon.

x Value	Interval	Bits Needed to Encode
37	1-64	6
49	37-64	5
39	37-49	4
39	39-49	4
41	39-49	4
42	41-49	4
42	42-49	3
47	42-49	3
48	47-49	2

The procedure that actually implements the shuffling for an x-reorderable polygon is given by the following steps, where the vertex numbering is cyclic (vertex 0 means vertex n, vertex n+1 means vertex 1, etc.):

1. The first vertex is m, the one having the minimum x.

2. The second vertex is r, the one having the maximum x.

3. If r = m+1, there are no "right side" reports; list the remaining vertices in the order m-1, m-2, ··· , r+2, r+1; stop.

4. If r = m-1, there are no "left side" reports; list the remaining vertices in the order m+1, m+2, ··· , r-2, r-1; stop.

5. Set p = m+1, q = m-1.

6. Compare x_p versus x_q.

7. If $x_p \leq x_q$,

   list p as the next vertex, and increment p.

   Else list q as the next vertex, and decrement q.

8. Return to step 6 if less than n vertices listed so far.

The information that must be encoded for each polygon, and the order in which it is encoded in the message, is as follows:

1. Its category — type of fillable (if either) and type of reorderable (if either).

2. The number n of its vertices.

3. The list of coordinate pairs x_i, y_i of its vertices, with the vertices listed clockwise if the polygon is non-reorderable or shuffled as above if it is reorderable.

The first piece of information requires 1 bit if the polygon is non-fillable, or 3 bits if it is fillable, with the specific encoding logic provided in Section 7.6.

The size of the field required to specify n, if fixed in size, would have to be large enough to handle the maximum possible number of vertices. Rather than waste that many bits, the field has been defined to be of variable size; specifically, n is specified by a series of 4 bit chunks, each of which incrementally encodes as many as 15 additional vertices. The number of chunks c required then becomes:

$$c = \frac{(n-2) + 1}{15} \quad \text{rounded up to the next higher integer}$$

where the -2 indicates that the first 2 vertices, representing the minimum legal polygon, are "free."

Finally, the vertex list consists of alternate x and y coordinates ordered according to the polygon reorderable type. If non-reorderable, each vertex coordinate x or y requires 6 bits; if reorderable, the number of bits for each reordered coordinate will start at 6 and possibly drop to as low as 0, while the other coordinate will always require the full 6 bits.

The complete detailed description of the polygon encoding scheme, for all cases and types of polygons, is provided by inference in Section 7.6, in which the bit-by-bit polygon decoding logic is specified.

7.6 POLYGON DECODING

The form of the polygon decoding algorithm, which is tasked to read the encoded bit stream and produce the clockwise ordered list of polygon vertices, depends upon whether or not the polygon was reorderable. The first steps of the decoding process make that determination:

1. Read V, the value of the first message bit.
2. If V=1, the polygon is fillable; if V=0, it is not fillable, stop.
3. If the polygon is fillable, read V, the value of the next 2 message bits.
4. The polygon type is specified by V as follows:

V	Fillable Type	Reorderable Type
0	x	neither
1	x	x
2	y	neither
3	y	y

The next steps of the decoding process read n, the number of vertices of the polygon:

1. Set n = 2 (the minimum polygon).
2. Read V, the value of the next 4 message bits.
3. Increase n by V.
4. If V < 15, stop; else if V = 15, return to step 2.

If the polygon was found to be non-reorderable, the reading of the vertex coordinates is straightforward and in the proper clockwise order:

1. Set i = 1.
2. Read V, the value of the next 6 message bits; set x_i = V+1.
3. Read V, the value of the next 6 message bits; set y_i = V+1.
4. Increment i; if i < n, return to step 2, else stop.

On the other hand, if the polygon was found to be reorderable, the order of the vertices must be unshuffled from the order they are read. Letting v and w be temporary arrays used to store the coordinates until the unshuffling has been resolved, the steps for an x-reorderable polygon become:

1. Read V, the value of the next 6 message bits; set x_1 = V+1.
2. Read V, the value of the next 6 message bits; set y_1 = V+1.
3. Compute b = $\log_2(64-x_1+1)$ rounded up to next integer.
4. If b>0,

   read V, the value of the next b bits; set $v_2 = x_1 + V$.

Else set $v_2 = x_1$.

5. Read V, the value of the next 6 message bits; set $w_2 = V+1$.
6. If $n = 2$, set $x_2 = v_2$, $y_2 = w_2$, and stop.
7. Compute $b = \log_2(v_2-x_1+1)$ rounded up to next integer.
8. If $b>0$,

   read V, the value of the next b bits; set $v_3 = x_1 + V$.

   Else set $v_3 = x_1$.
9. Read V, the value of the next 6 message bits; set $w_3 = V+1$.
10. Set $i = 4$.
11. While $i \leq n$:

    Compute $b = \log_2(v_2-v_{i-1}+1)$ rounded up to next integer.

    If $b>0$,

      Read V, the value of the next b bits; set $v_i = v_{i-1} + V$.

    Else set $v_i = v_{i-1}$.

      Read V, the value of the next 6 message bits; set $w_i = V+1$
12. Compute s_0, the reference slope, defined by

$$s_0 = \frac{w_2 - y_1}{v_2 - x_1}$$

13. Set $p = 2$, $q = n$, $i = 3$.
14. While $i \leq n$:

    Compute s, the slope for temporary vertex i, defined by

$$s = \frac{w_i - y_1}{v_i - x_1}$$

    If $s > s_0$:

      set $x_p = v_i$, $y_p = w_i$

      increment p

    Else if $s < s_0$:

      set $x_q = v_i$, $y_q = w_i$

      decrement q
15. Set $x_p = v_2$, $y_p = w_2$.

The steps for a y-reorderable polygon are just the converse of these, that is, with x and y interchanged.

7.7 POLYGON-FILL PROCEDURE

The purpose of the polygon-fill routine is to identify all points on the map that lie within the boundary of the polygon specified by the input list of vertices. For "normal" convex polygons, this process is extremely simple. However, when concave vertices exist, or worse yet when the boundary lines of the polygon touch or even cross each other, determining just where the "inside" and the "outside" of the polygon lie can be quite complex. The initial polygon-construction routine, which traces the outside of the weather region, can never produce crossing lines. However, the vertex-reduction routine can, in some rare cases because of approximations and integer roundoffs, produce this effect.

The polygon-fill routine contains two different fill algorithms: a simple one that can handle only fillable polygons (as defined above) and a general one that is applicable to all cases. Since the simple routine executes much faster than the general-case routine and many real weather regions qualify as fillable, the inclusion of the additional code required by the special-case routine is justified.

The polygon-fill routine in either the simple or the general case has two steps: drawing the boundary lines, and finding the points that lie within this boundary. The procedure for drawing the boundary lines is common to both routines; the dichotomy is in the complexity of determining inside from outside.

A polygon with n vertices also has n boundary lines, each extending from one vertex to the next. For the i^{th} boundary line let:

$$\Delta x = x_{i+1} - x_i = \text{sign}(\Delta x) * |\Delta x| = s_x * dx$$

$$\Delta y = y_{i+1} - y_i = \text{sign}(\Delta y) * |\Delta y| = s_y * dy$$

Then the map cells r,c (row,column) on the boundary line are found according to the magnitude of the slope of the line as indicated in the following cases:

1. dx = 0: (horizontal line)

   $r = x_i,\ c = y_i + n * s_y \quad n=0,dy$

2. dy = 0: (vertical line)

   $c = y_i,\ r = x_i + n * s_x \quad n=0,dx$

3. dx = dy: (diagonal line)

   $r, c = x_i + n * s_x,\ y_i + n * s_y \quad n=0,dx$

4. dx > dy: (x-oblique line)

   $$r, c = x_i + n * s_x,\ y_i + n * s_y * \frac{dy}{dx} \quad n=0,dx$$

   c rounded to nearest integer

5. dy > dx: (y-oblique line)

   $$r, c = x_i + n * s_x * \frac{dx}{dy},\ y_i + n * s_y \quad n=0,dy$$

   r rounded to nearest integer

In the two oblique cases, one coordinate has a smaller delta than the other. Therefore, the coordinate with the smaller magnitude delta must repeat its value. For example, a sample y-oblique line would appear as shown in Figure 7-11.

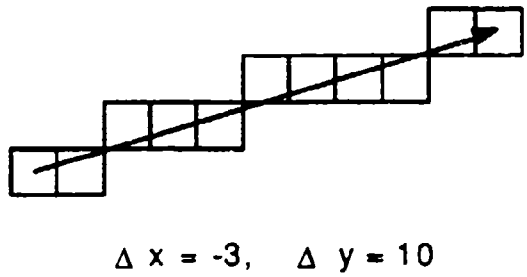


Figure 7-11. Example of Y-Oblique Line.

Once all the points comprising the polygon boundary lines have been determined, the interior fill procedure for the simple algorithm is extremely straightforward. By the definition of an x-fillable polygon, the filled region for a given map row, if any exists, always extends from the minimum boundary line point on that row to the maximum boundary line point on the row. Thus, the complete fill algorithm for the simple case consists of the following steps:

1. Initialize the minimum and maximum points for each row r to $s_r = L$ (a large number) and $b_r = -1$, respectively.
2. Draw all the boundary lines of the polygon point by point. For each point r,c (row,column) on each boundary line, redefine b_r and s_r by:

   $b_r = \text{Max}\ \{b_r, r\}$

   $s_r = \text{Min}\ \{s_r, r\}$
3. For each row r of the map, the filled region (if any) consists of the points c, $s_r \leq c \leq b_r$.

For a y-fillable polygon, just reverse row and column and proceed in a similar manner.

When the polygon is not simple, a given row of the map may switch from "outside" to "inside" and back any number of times. The general rule is that every time a boundary line of the polygon is crossed, such a switch occurs. Three factors complicate the employment of this apparently straightforward rule: an end point of a boundary line is part of two lines; a y-oblique line has two or more points in a row that are part of that same line; and two or more lines can cross, so that the crossing point is part of both lines.

Thus, it is clear that each boundary line point must be labeled with a tag indicating the number of lines that are crossed when that point is encountered. With this information, the fill algorithm for row r would know that when it encounters a boundary point labeled with an odd number it should switch from being inside the polygon and filling points to being outside, or vice versa; when it encounters a boundary point with an even label it should continue unchanged.

Even though it is easy to determine if a region changes from inside to outside when an intermediate point on a boundary line is crossed, there is some difficulty with end points. This difficulty occurs in determining the number of lines being crossed at a boundary point when one of the three complicating factors above occurs. First consider the end point of a boundary line. Three cases can exist (see Figure 7-12). In the reverse joining case, the elbow point must be labeled with 0 crossings, as no switch from outside to inside has occurred on that row. In the continue joining case on the other hand, a label of 1 is required as a switch does occur. Finally, in the flat joining case, the correct action can only be determined by examining the next elbow point to see whether an "extended reverse" or "extended continue" joining has occurred. Then the leftmost of the two elbow points is labeled according to the previous rule and the second elbow point is labeled with a 0 so as not to undo that action.

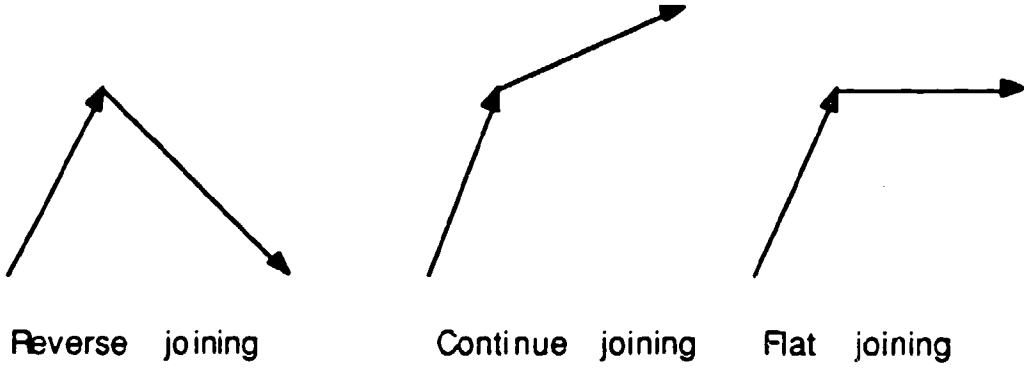


Figure 7-12. Boundary Point Joining Cases.

Next consider the problem with y-oblique lines. Although two or more points of the line can exist on the same row r, only one line is being crossed. Thus, the labeling rule to follow is that only the leftmost of the points is to be labeled; other points of the line in the same row, if any, are ignored.

Finally, the possibility of a single map point being part of two or more boundary lines because of line convergence or crossings is easily accommodated. Each point's label is initialized to 0 crossings; each time it is found to be part of a boundary line, the label is incremented or not according to the earlier rules. The final label will then properly indicate the number of crossings at the point.

An example of the label values on the boundary points of a complex polygon, illustrating the above cases, is shown in Figure 7-13.

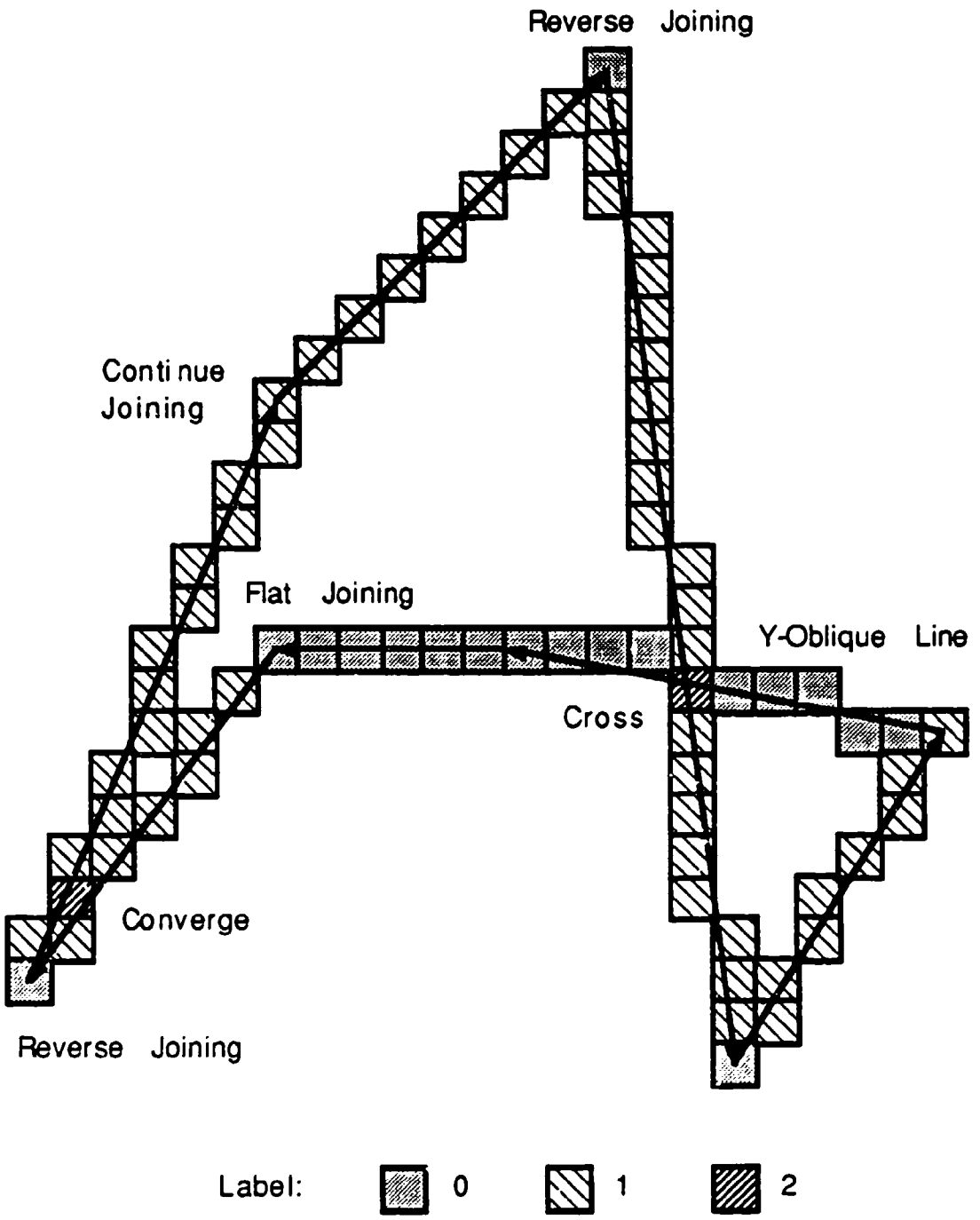


Figure 7-13. Labeling a Complex Polygon.

Incorporating these rules for boundary-point labeling, the general fill algorithm can be expressed as follows:

1. Initialize the label for every point r,c (row,column) to 0.
2. Draw all the boundary lines of the polygon point by point. For each point r,c on each boundary line, decide whether or not to increment the point's label according to the following rules:
   a. Increment the first end point of a line if the end point implements a continue joining or if the end point is the left end of an extended continue joining.
   b. Increment each interior point of a line unless the point is on a horizontal or y-oblique line and the point to its left is also on the line.
   c. Do not increment the second end point of a line (it will be handled as the first end point of the next line).
3. For each row, initialize the case to "outside."
4. Scan each row from left to right. When an odd-valued labeled point is found, switch between "outside" and "inside." Fill all points when "inside" is true.

One modification to these rules is introduced to account for the presence of the "tunnels" through the polygon created as part of the hole-preservation algorithm described in Section 4. The new rule is that whenever the sequence "fill, no-fill, fill" is encountered in a row, the middle point is also filled in, thereby removing the vertical tunnels from view.

8. QUADRILATERAL-REPRESENTATION ALGORITHM

Should the input weather map be extremely complex, or the bit limitation be especially severe, it is possible that some of the lower-priority weather levels will not be allocated enough bits to separately represent their weather regions. In that case, the default minimum representation has been defined to be a single quadrilateral that encompasses all the level's weather regions.

8.1 QUADRILATERAL GENERATION

The method chosen to construct this quadrilateral is to generate a set of vertices such that the polygon they define includes all the weather regions on the given single-level map. This set of vertices is then passed to the normal polygon reduction routine described in Section 7 with the command that the number of vertices be cut to four. The result will be the quadrilateral that best balances representation of true weather pixels with minimization of added overfill pixels.

An example of the type of initial polygon we desire to construct, and the resultant quadrilateral, is seen in Figure 8-1.

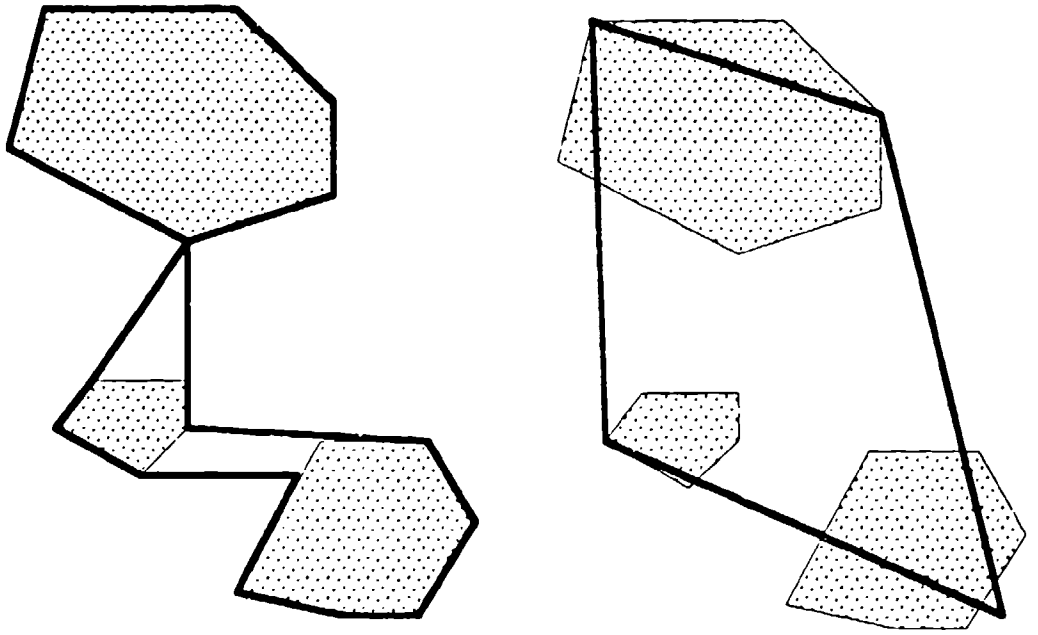


Figure 8-1. Conversion of Initial Polygon Contour to Final Quadrilateral.

In particular, a vertex of the initial polygon is to be generated at each end of any map row that contains non-zero pixel values. The map-preparation routine has already calculated these end

values for each row i of level L as $min_{L,i}$ and $max_{L,i}$, with $min_{L,i} > 64$ used to indicate that row i has only zero pixel values. Thus, the clockwise vertex generation algorithm becomes:

1. Initialize the row number to r = 1.
2. If non-zero pixel values exist on row r, add a vertex at $(r, max_{L,i})$
3. Increment r; if $r \leq 64$, return to step 2.
4. Initialize the row number to r = 64.
5. If non-zero pixel values exist on row r, add a vertex at $(r, min_{L,i})$
6. Decrement r; if $r \geq 1$, return to step 5.

The polygon-reduction routine also needs to know the area of the region encompassed by this polygon. Using the trapezoidal-area formula, each row i having non-zero pixel values, after the first such row, adds to the area an amount:

$$\Delta area = \frac{1}{2} * \{ (max_i - min_i) + (max_k - min_k) \} * (i - k)$$

where k is the previous row having non-zero pixel values. In the default case of only one row i having non-zero pixel values,

$$area = max_i - min_i$$

8.2 QUADRILATERAL ENCODING

The quadrilateral is not considered to be a separate shape; rather, it is treated as just a polygon that happens to always have 4 vertices. Thus, the encoding scheme for a quadrilateral is exactly the same as presented in Section 7 for polygons. In particular, it must be classified as fillable or not, and reorderable or not, and the proper encoding method utilized.

8.3 QUADRILATERAL DECODING

Once again, a quadrilateral is treated exactly the same as any other polygon by the decoder. In fact, the decoder won't be able to determine whether a 4-vertex polygon is simply representing a single weather region on the level or the entire weather level. Depending upon whether the quadrilateral is fillable or not, either of the fill procedures of Section 7 could be employed for it.

APPENDIX

POLYGON-ELLIPSE USER-SETTABLE PARAMETERS

This Appendix lists and defines each of the user-settable parameters presently existing in the PE software. It also lists the default value of each parameter, and discusses the performance effects of changes to other legal values.

Map Bit Limit (b_limm) Default = 1280

This parameter is the maximum allowable number of bits that can be used to encode the weather map. Changing the setting will increase or decrease the distortion produced in the map representation.

Level Renumbering (remap_flag) Default = 0

This parameter specifies whether the standard weather level numbering scheme should be revised. The meanings of the possible values are:

0 = no renumbering of weather levels

1 = renumber the weather levels — the user must enter in order the new numbers for levels 1 through 6 into remap[6].

The remapping feature is useful when fewer than 7 levels of weather are desired on the output display. For example, a 3-level display — no weather, rain, storms — would be produced by entering 1 followed by the 6 values 1,1,2,2,2,2.

Level Priority Ordering (levtype) Default = 0

This parameter specifies the type of level priority ordering to be used in the encoding logic when distortion must be introduced. The meanings of the possible values are:

0 = "normal" — level 1 lowest priority, level 6 highest priority

1 = "reversed" — level 6 lowest priority, level 1 highest priority

2 = "user-ordered" — the user must enter the 6 levels in order from lowest priority to highest priority into in_maplev[6].

When distortion is required, the levels are degraded in order from lowest to highest priority.

Minimum Region Size (thresh[6]) Default = 6,3,1,1,1,1

These parameters specify, for each level 1 through 6, the smallest weather region that will be encoded. If the values are decreased, more weather "spots" will be maintained on the output map; if they are increased, the larger regions will be more faithfully represented as more bits are freed up. These thresholds also apply to minimum-size hole representations on the levels, except that no hole of size less than 6 will ever be encoded.

Highest Hole Level (high_hole) Default = 6

This parameter specifies the highest level for which holes in weather regions (areas of less severe weather within a weather region) will be maintained by the encoding algorithm. By reducing

the value, holes in severe weather regions will be ignored; this will free up more bits for accurate representation of weather regions.

Initial Distortion (init_dist) Default = 0%

This parameter specifies how much polygon distortion will be accepted on the first pass through the encoding algorithm. If set to 0, an attempt at the best possible representation will be made; if set higher, processing time will be saved for complex maps as fewer iterations through the distortion sequence (see Section 3) will be required before the bit limit is satisfied.

Polygon Angle Breaks (a_small, o_small) Default = 20°, 30°

These parameters control when, in the tracing of the line segments in the contour of a weather region (see Section 7), a new polygon side is initiated. As long as the newest segment direction differs by less than a_small degrees from the present direction of the current side, and less than o_small degrees from the initial direction of the current side, the segment is added to the current side to produce a longer side. Decreasing these values will produce more accurate polygon representations for simple maps, but more region distortion and more processing time for complex maps.

REFERENCES

[1] Orlando, V. A. and P.R. Drouilhet, "Mode S Beacon System: Functional Description." M.I.T. Lincoln Laboratory, ATC-42, Revision D, (August 29, 1986).

[2] Lelewer, D. E. and D. S. Hirschberg, "Data Compression," *ACM Computing Surveys* Vol. 19, No. 3 (September 1987): 261-296.

[2] Huffman D. A., "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings IRE*, Vol. 40, No. 9 (September 1952): 1098-1101.

[3] Gertz, J., "The Weather Huffman Method for Data Compression of Weather Maps" report in progress.

[4] Winston, P. J. and B. K. P. Horn, "LISP," Second Edition, Reading, MA: Addison-Wesley Publishing Company, 1984, Chapter 10.